Algorithms
Collection

Free ebook by
Adam Higherstein

Algorithms
Collection

Free ebook by
Adam Higherstein

Main contents

- Approximations
- Biggest of 3
- Bit operations
- Dijkstra path finding
- Bin packing
- Pascal triangle
- Recursive functions
- Statistics
- Ford-Fulkerson path finding
- DSP & FFT
- Insertion sort
- Quick sort
- Shell sort
- Selection sort

Approximation of PI

Approximations

PI

Help functions

```
double fact(int k)
{
    double f = 1;
    int i;
    for (i = 1; i <= k; i++)
        f *= i;

        return f;
}
```

Approximations

PI

Help functions

```
double power(double base, int k)
{
    double p  = 1;
    int i;
    for (i = 1; i <= k; i++)
     p *= base;

      return p;
}
```

Approximations

PI

Madhava de Sangamagrama

(1350-1425)

$$\pi = \sqrt{12}\sum_{i=0}^{\infty}\frac{(-1)^i}{(2i+1)3^i}$$

$$\pi = \sqrt{12}\left(1 - \frac{1}{3\cdot 3} + \frac{1}{5\cdot 3^2} - \frac{1}{7\cdot 3^3} + \cdots\right)$$

https://alchetron.com/Madhava-of-Sangamagrama

Approximations

Madhava de Sangamagrama

(1350-1425)

$$\pi = \sqrt{12}\sum_{i=0}^{\infty}\frac{(-1)^i}{(2i+1)3^i}$$

$$\pi = \sqrt{12}\left(1 - \frac{1}{3\cdot 3} + \frac{1}{5\cdot 3^2} - \frac{1}{7\cdot 3^3} + \cdots\right)$$

https://alchetron.com/Madhava-of-Sangamagrama

PI

```
double pi1,pi2,pi3;

pi1 = 0;
int last_member = 10;
int i;

for (i = 0; i <= last_member; i++)
{
    pi1 = pi1 +  power(-1.0/3, i)/(2 * i + 1);
}
pi1 = sqrt(12)* pi1;

printf("Pi 1 is %lf \n", pi1);
```

Approximations

Madhava de Sangamagrama

(1350-1425)

$$\pi = \sqrt{12}\sum_{i=0}^{\infty}\frac{(-1)^i}{(2i+1)3^i}$$

$$\pi = \sqrt{12}\left(1 - \frac{1}{3\cdot 3} + \frac{1}{5\cdot 3^2} - \frac{1}{7\cdot 3^3} + \cdots\right)$$

https://alchetron.com/Madhava-of-Sangamagrama

PI

```
double pi1,pi2,pi3;

pi1 = 0;
int last_member = 10;
int i;

for (i = 0; i <= last_member; i++)
{
    pi1 = pi1 +  power(-1.0/3, i)/(2 * i + 1);
}
pi1 = sqrt(12)* pi1;

printf("Pi 1 is %lf \n", pi1);
```

```
Pi 1 is 3.141593
```

Approximations

PI

Newton:

$$\frac{\pi}{2} = \sum_{k=0}^{\infty} \frac{k!}{(2k+1)!!} = \sum_{k=0}^{\infty} \frac{2^k k!^2}{(2k+1)!} = 1 + \frac{1}{3}\left(1 + \frac{2}{5}\left(1 + \frac{3}{7}(1 + \cdots)\right)\right)$$

Approximations

PI

Newton:

$$\frac{\pi}{2} = \sum_{k=0}^{\infty} \frac{k!}{(2k+1)!!} = \sum_{k=0}^{\infty} \frac{2^k k!^2}{(2k+1)!} = 1 + \frac{1}{3}\left(1 + \frac{2}{5}\left(1 + \frac{3}{7}(1 + \cdots)\right)\right)$$

```
pi2 = 0;
for (i = 0; i <= last_member; i++)
{
    pi2 = pi2 + (power(2,i)*fact(i)*fact(i))/fact(2*i + 1);

}
pi2 *= 2;
printf("Pi 2 is %lf \n", pi2);
```

Approximations

PI

Newton:

$$\frac{\pi}{2} = \sum_{k=0}^{\infty} \frac{k!}{(2k+1)!!} = \sum_{k=0}^{\infty} \frac{2^k k!^2}{(2k+1)!} = 1 + \frac{1}{3}\left(1 + \frac{2}{5}\left(1 + \frac{3}{7}(1 + \cdots)\right)\right)$$

```
pi2 = 0;
for (i = 0; i <= last_member; i++)
{
    pi2 = pi2 + (power(2,i)*fact(i)*fact(i))/fact(2*i + 1);

}
pi2 *= 2;
printf("Pi 2 is %lf \n", pi2);
```

```
Pi 2 is 3.141106
```

Approximations

PI

Ramanujan:

$$\frac{1}{\pi} = \frac{2\sqrt{2}}{9801} \sum_{k=0}^{\infty} \frac{(4k)!(1103 + 26390k)}{(k!)^4 396^{4k}}$$

Approximations

PI

Ramanujan:

$$\frac{1}{\pi} = \frac{2\sqrt{2}}{9801} \sum_{k=0}^{\infty} \frac{(4k)!(1103 + 26390k)}{(k!)^4 396^{4k}}$$

```
pi3 = 0;
for (i = 0; i <= last_member; i++)
{
    pi3 = pi3 + (fact(4*i)*(1103 + 26390*i))/(fact(i)*fact(i)*fact(i)*fact(i)*power(396,4*i));

}
pi3 = 2*sqrt(2)/9801 * pi3;
pi3 = 1/pi3;

printf("Pi 3 is %lf \n", pi3);
```

Approximations

PI

Ramanujan:

$$\frac{1}{\pi} = \frac{2\sqrt{2}}{9801} \sum_{k=0}^{\infty} \frac{(4k)!(1103 + 26390k)}{(k!)^4 396^{4k}}$$

```
pi3 = 0;
for (i = 0; i <= last_member; i++)
{
    pi3 = pi3 + (fact(4*i)*(1103 + 26390*i))/(fact(i)*fact(i)*fact(i)*fact(i)*power(396,4*i));

}
pi3 = 2*sqrt(2)/9801 * pi3;
pi3 = 1/pi3;

printf("Pi 3 is %lf \n", pi3);
```

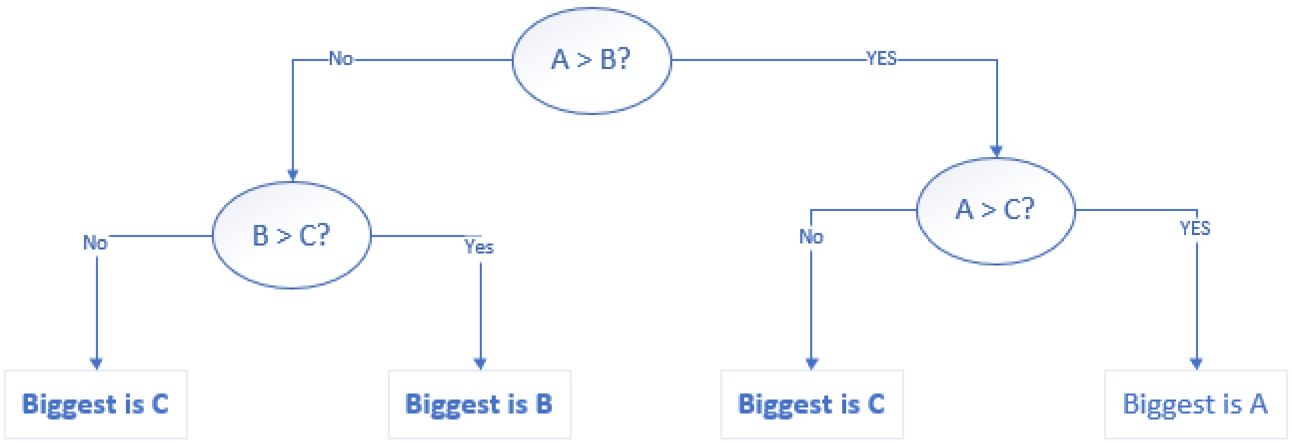
```
Pi 3 is 3.141593
```

Biggest of 3

Kakelino

Biggest of 3 values?

Decision tree?



Biggest of 3

Biggest of 3 values?
Way 1

Kakelino

```
int A,B,C;
A = 10; B = 20; C = 30;
if (A > B)
    if (A > C)
        printf("Biggest is A, %d \n", A);
    else
        printf("Biggest is C, %d \n", C);
else
    if (B > C)
        printf("Biggest is B, %d \n", B);
    else
        printf("Biggest is C, %d \n", C);
```

```
Biggest is C, 30
```

Kakelino

Biggest of 3 values?
Way 2

```
int A,B,C;
A = 10; B = 20; C = 30;
if (A > B && A > C)
    printf("Biggest is A, %d\n", A);
else
    if (B > A && B > C)
        printf("Biggest is B, %d\n", B);
else
    printf("Biggest is C, %d\n", C);
```

```
Biggest is C, 30
```

Kakelino

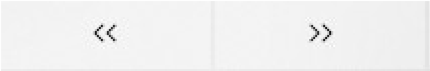
Biggest of 3 values?
Way 3

```
int A,B,C;
A = 10; B = 20; C = 30;
int max = A;
if (B > max)
  max = B;
if (C > max)
    max = C;

printf("Biggest values is %d\n", max);
```

```
Biggest values is 30
```

Bit operations

Bit operations



Bit operations

```
/* bit operations
AND &
OR |
XOR ^
shift << >>
*/

int a = 188;     // 10111100
int b = 211;     // 11010011
```

Bit operations

```c
/* bit operations
AND &
OR |
XOR ^
shift << >>
*/

int a = 188;    // 10111100
int b = 211;    // 11010011
```

```c
/*
a & b
10111100
11010011
10010000    ==> 144
*/

printf("a & b is %d \n", a & b);
```

```
a & b is 144
```

Bit operations

```
/* bit operations
AND &
OR |
XOR ^
shift << >>
*/

int a = 188;    // 10111100
int b = 211;    // 11010011
```

```
/*
a | b
10111100
11010011
11111111   => 255
*/

printf("a | b is %d \n", a | b);
```

```
a | b is 255
```

Bit operations

```c
/* bit operations
AND &
OR |
XOR ^
shift << >>
*/

int a = 188;     // 10111100
int b = 211;     // 11010011
```

```c
/*
a ^ b
10111100
11010011
01101111    ==> 111
*/

printf("a ^ b is %d \n", a ^ b);
```

```
a ^ b is 111
```

Bit operations

```c
/* bit operations
AND &
OR |
XOR ^
shift << >>
*/

int a = 188;     // 10111100
int b = 211;     // 11010011
```

```c
/*
 a << 2
 10111100
 1011110000    ==> 752
*/

 printf("a << 2 is %d \n", a << 2);
```

```
a << 2 is 752
```

Bit operations

```c
/* bit operations
AND &
OR |
XOR ^
shift << >>
*/

int a = 188;    // 10111100
int b = 211;    // 11010011
```

```c
/*
 b >> 3
 11010011
 00011010    ==> 26
*/

 printf("b >> 3 is %d \n", b >> 3);
```

```
b >> 3 is 26
```

Bit operations

Checking the state of a bit

```
/* checking the state of a spesific bit:
1) shift bit queue to the left until the goal bit is lsb (0. bit)
2) take AND with value 1   (can be presented also as e.g. 00000001)
3) result is the state of the bit we wanted to check
```

Bit operations

```
/* checking the state of a spesific bit:
1) shift bit queue to the left until the goal bit is lsb (0. bit)
2) take AND with value 1   (can be presented also as e.g. 00000001)
3) result is the state of the bit we wanted to check
```

Checking the state of a bit

```
example:
a is our queue
10111100
we want to check the 3. bit  (if we start from position 0, it is really 2. bit)
we can see that bit state is 1
shift queue now 2 times to the left:
we get
00101111
take value 1 with
00101111
00000001
Take AND
00101111
00000001
00000001
SO, the state i1 1.
```

Bit operations

```
/* checking the state of a spesific bit:
1) shift bit queue to the left until the goal bit is lsb (0. bit)
2) take AND with value 1   (can be presented also as e.g. 00000001)
3) result is the state of the bit we wanted to check
```

Checking the state of a bit

```
example:
a is our queue
10111100
we want to check the 3. bit  (if we start from position 0, it is really 2. bit)
we can see that bit state is 1
shift queue now 2 times to the left:
we get
00101111
take value 1 with
00101111
00000001
Take AND
00101111
00000001
00000001
SO, the state i1 1.
```

```
The state of the 2. bit is 1
```

Bit operations

Inverting a bit

```
/* Toggling (inverting) a bit
1) create a mask (special bit queue) where it is 1
   in the position that is to be inverted of the original
   bit queueu.
2) take XOR with the mask and original queue
3)  original bit queue is replaced by the result of the operation
Example:
b is 11010011
we want to invert the 4. bit (starting from index 0, it is 3.)
we get the mask by shifting value 1 to the right 3 times
we get   00001000
take XOR
11010011
00001000
11011011   ==> 219
*/
```

Bit operations

Inverting a bit

```
/* Toggling (inverting) a bit
1) create a mask (special bit queue) where it is 1
   in the position that is to be inverted of the original
   bit queueu.
2) take XOR with the mask and original queue
3)  original bit queue is replaced by the result of the operation
Example:
b is 11010011
we want to invert the 4. bit (starting from index 0, it is 3.)
we get the mask by shifting value 1 to the right 3 times
we get   00001000
take XOR
11010011
00001000
11011011   ==> 219
*/
```

```
int bitplace = 3;
int mask = 1 << bitplace;
b = b ^ mask;
printf("b is now %d \n", b);
```

```
b is now 219
```

Bit operations

If XOR is missing

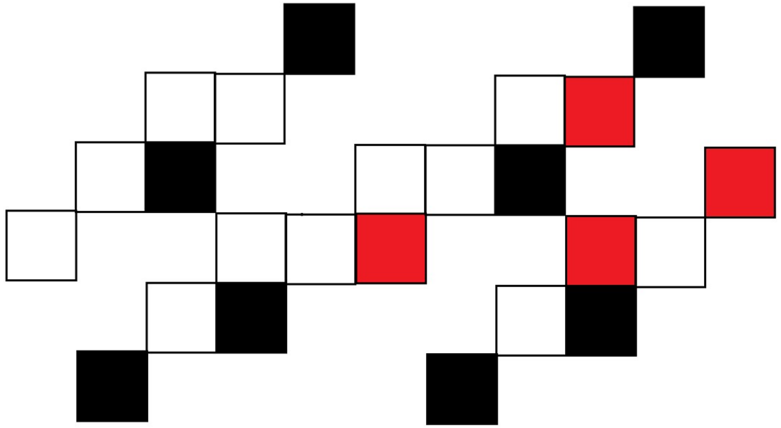
```
/* if XOR is missing?
  we can create XOR with or, ! and and
  x XOR y  = x OR y & !(x & y)

*/

int x = 100;
int y = 200;
int result = x ^ y;
printf("x XOR y is  %d \n", result );

result  = (x | y) & ~(x & y);

printf("x XOR y is  %d \n", result );
```

```
x XOR y is   172
x XOR y is   172
```

Try examples!

Check also 7 segment example!

Dijkstra

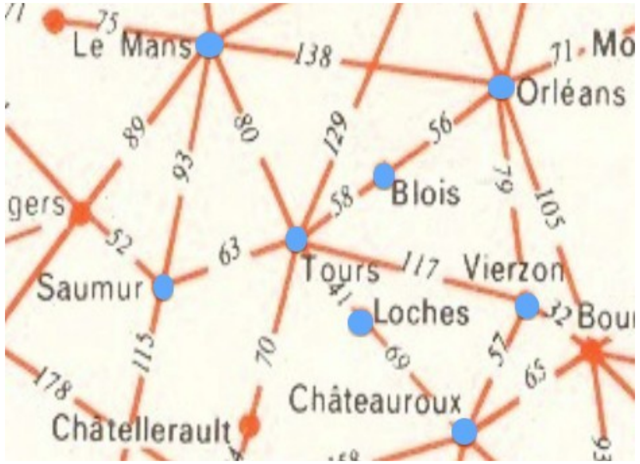


Edsger Dijkstra
shortest routes demo

Dijkstra Example
Shortest routes from Le Mans to other c

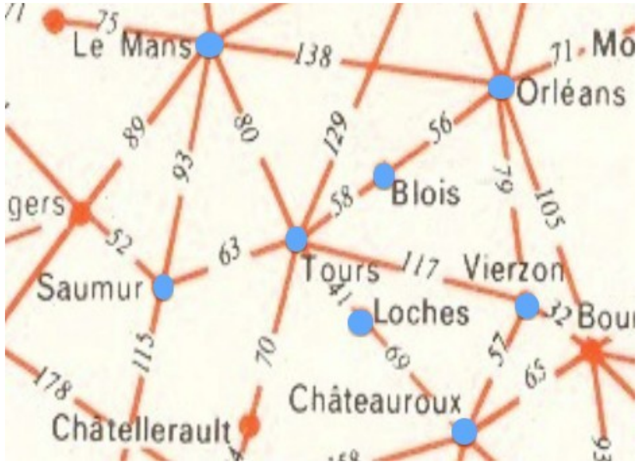
Map of France is here:

Edsger Dijkstra
shortest routes demo

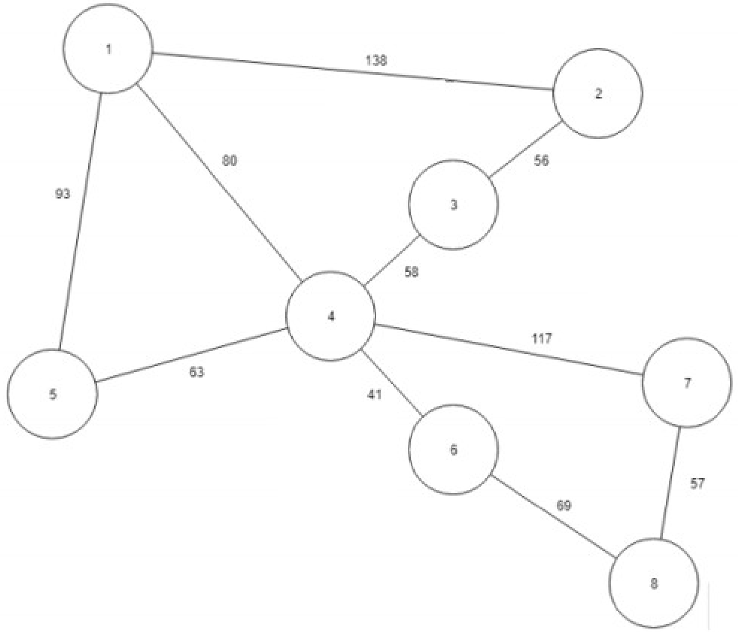


Blue circles are cities. We start from Le Mans.

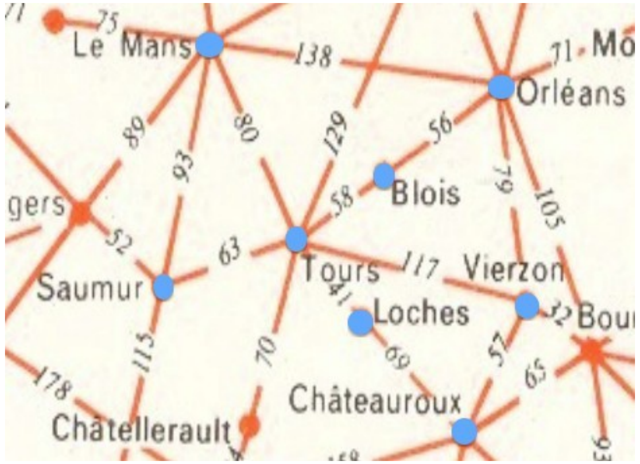
Edsger Dijkstra shortest routes demo



Here the network/graph as a diagram:



Edsger Dijkstra shortest routes demo



Here the network/graph as a matrix:

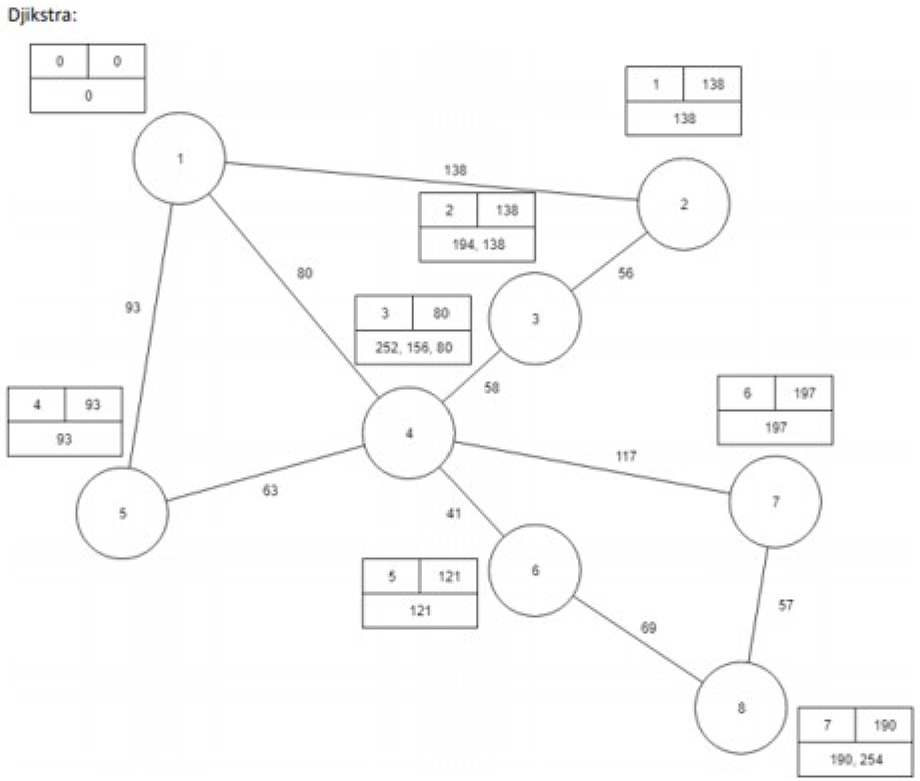
	1	2	3	4	5	6	7	8
1	0	138	INF	80	93	INF	INF	INF
2	138	0	56	INF	INF	INF	79	INF
3	INF	56	0	58	INF	INF	INF	INF
4	80	INF	58	0	63	41	117	INF
5	93	INF	INF	63	0	INF	INF	INF
6	INF	INF	INF	41	INF	0	INF	69
7	INF	79	INF	117	INF	INF	0	57
8	INF	INF	INF	INF	INF	69	57	0

Edsger Dijkstra shortest routes demo

Here is the solution matrix:
note priority queue

Round nr	Current node	Neighbours	Updates	Queue (priority queue)
1	1	2,4,5	2(true, 138, 1), 4(true, 80, 1),5(true,93,1)	4(true, 80, 1), 5(true, 93, 1), 2(true, 138, 1)
2	4 80	3,5,6,7	3(true,58,4) => 138 5(true,63,4) => 143 NO 6(true,41,4) => 121 7(true,117,4) =>197	5(true, 93, 1) 6(true,121,4) 2(true, 138, 1) 3(true,138,4) 7(true,197,4)
3	5 93	1, 4	1 NO 4 NO	6(true,121,4) 2(true, 138, 1) 3(true,138,4) 7(true,197,4)
4	6 121	4,8	4 NO 8(true, 121 + 69, 6)	2(true, 138, 1) 3(true,138,4) 8(true, 190, 6) 7(true,197,4)
5	2 138	1,3	1 NO 3(true, 138+56, true) NO	3(true,138,4) 8(true, 190, 6) 7(true,197,4)
6	3 138	2,4	2 NO 3 NO	8(true, 190, 6) 7(true,197,4)
7	8 190	6,7	7 NO 7 NO	7(true,197,4)
8	7 197	4,8	4 NO 8 NO	

Edsger Dijkstra
shortest routes demo

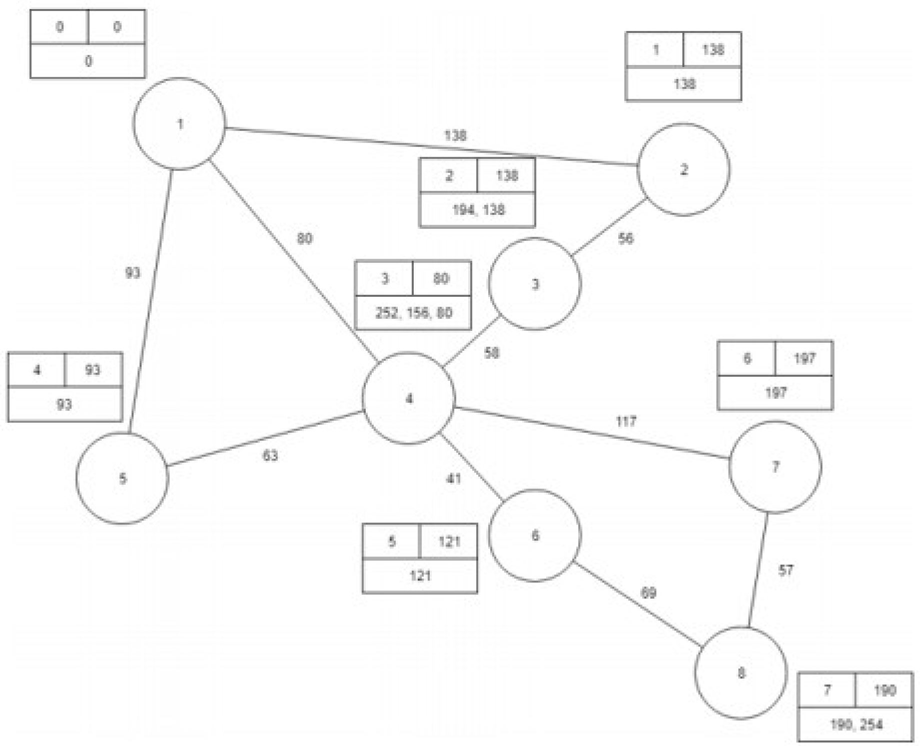


Edsger Dijkstra
shortest routes demo

Results of the code:

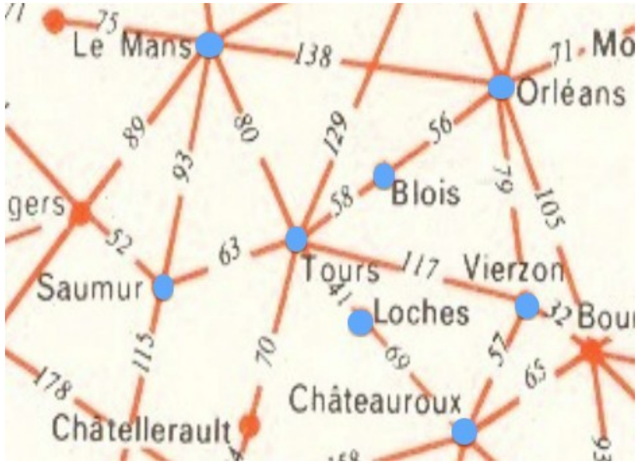
```
0..0 -> 0
0..1.. -> 138
0..3..2.. -> 138
0..3.. -> 80
0..4.. -> 93
0..3..5.. -> 121
0..3..6.. -> 197
0..3..5..7.. -> 190
```

Djikstra:



Edsger Dijkstra
shortest routes demo

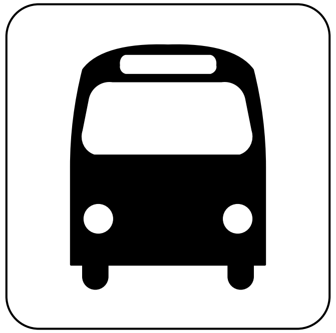
Try to simulate it!



Bin packing

First fit method

Bin packing



Travelling groups: whole group has to have room in a bus

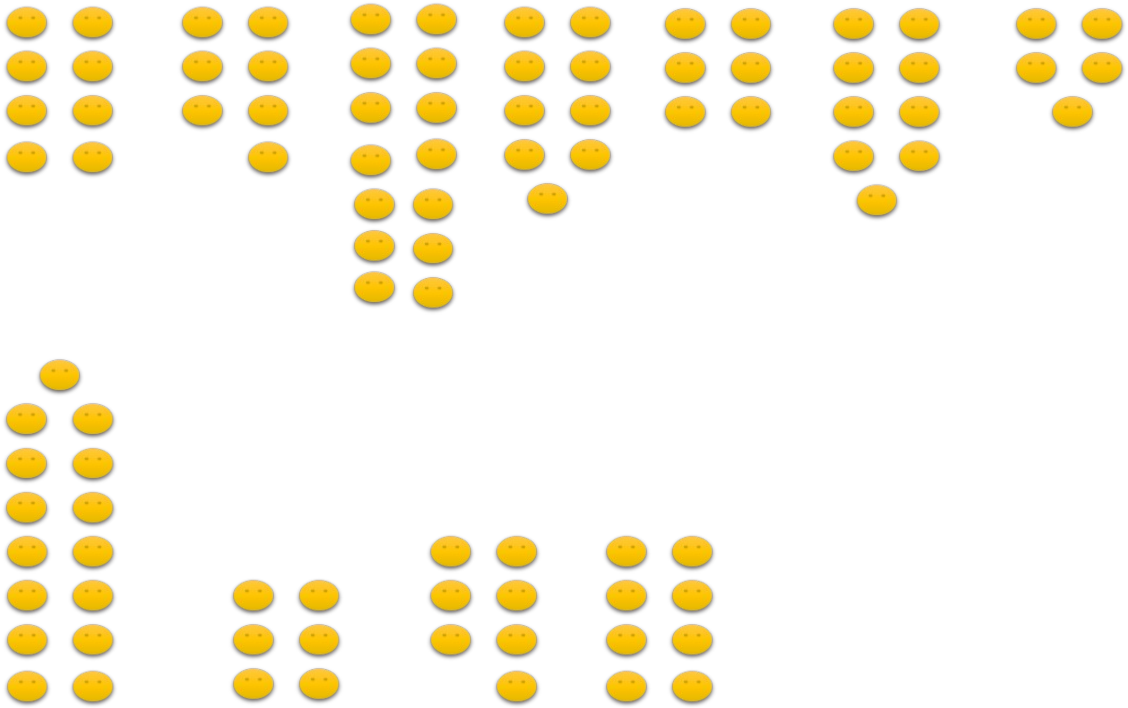
20 persons can take room in a bus

Code School

Bin packing

First fit method

Here are passenger groups
11 groups

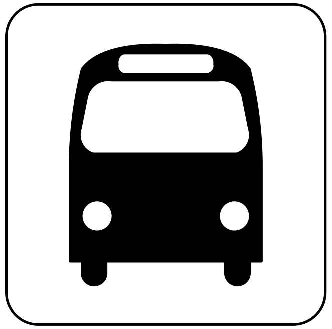


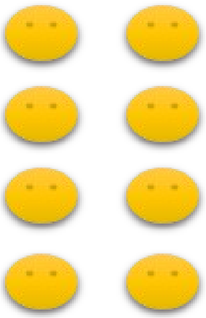
Code School

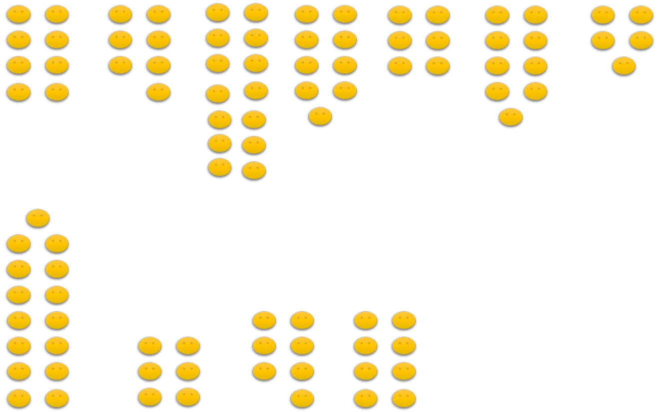
Bin packing

First fit method

First group has
8 persons:
put persons to bus 1





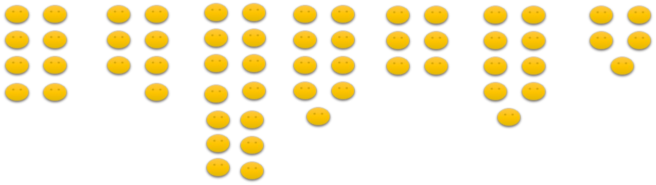


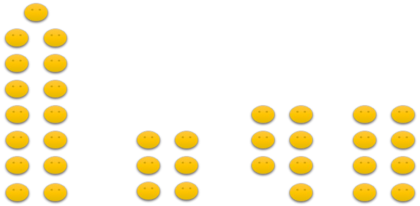
BUS 1

Code School

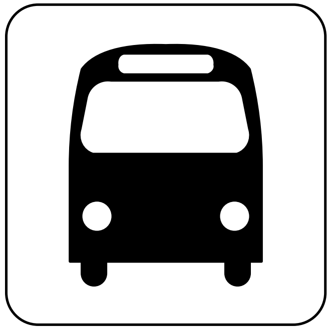
Bin packing

First fit method

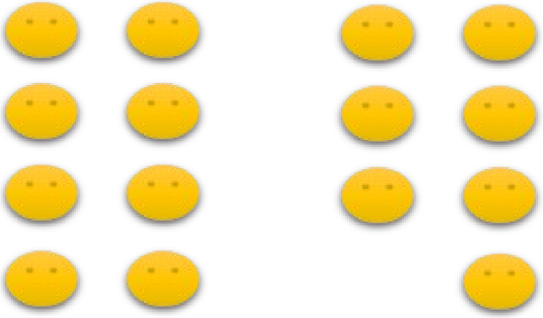




Second group has
7 persons:
put persons to bus 1, too



BUS 1

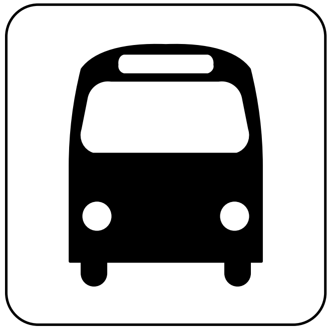


Code School

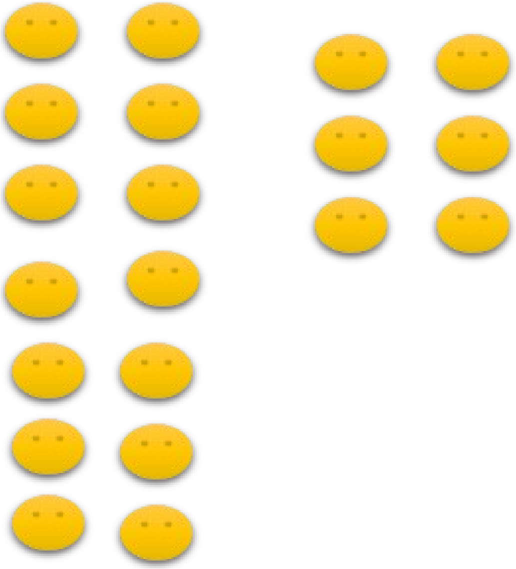
Bin packing

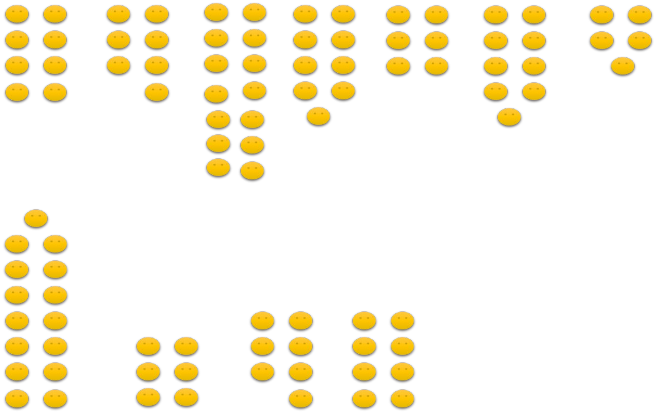
First fit method

3. group has
14 persons:
put persons to bus 2



BUS 2



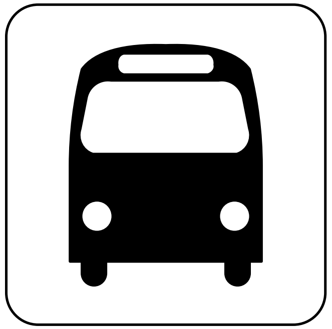


Code School

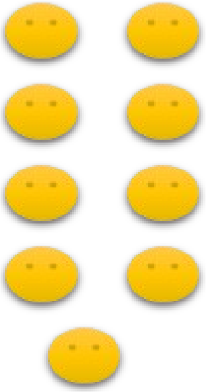
Bin packing

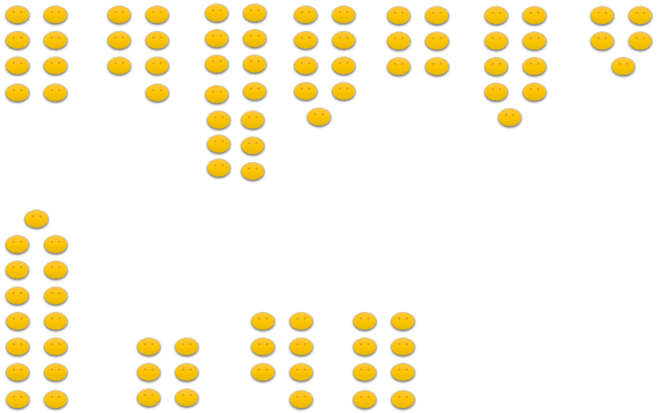
First fit method

4. group has
9 persons:
put persons to bus 3



BUS 3



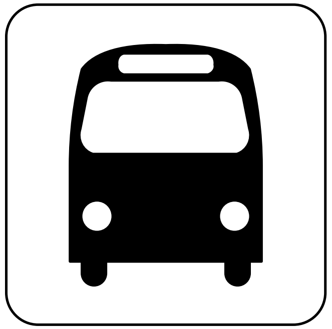


Code School

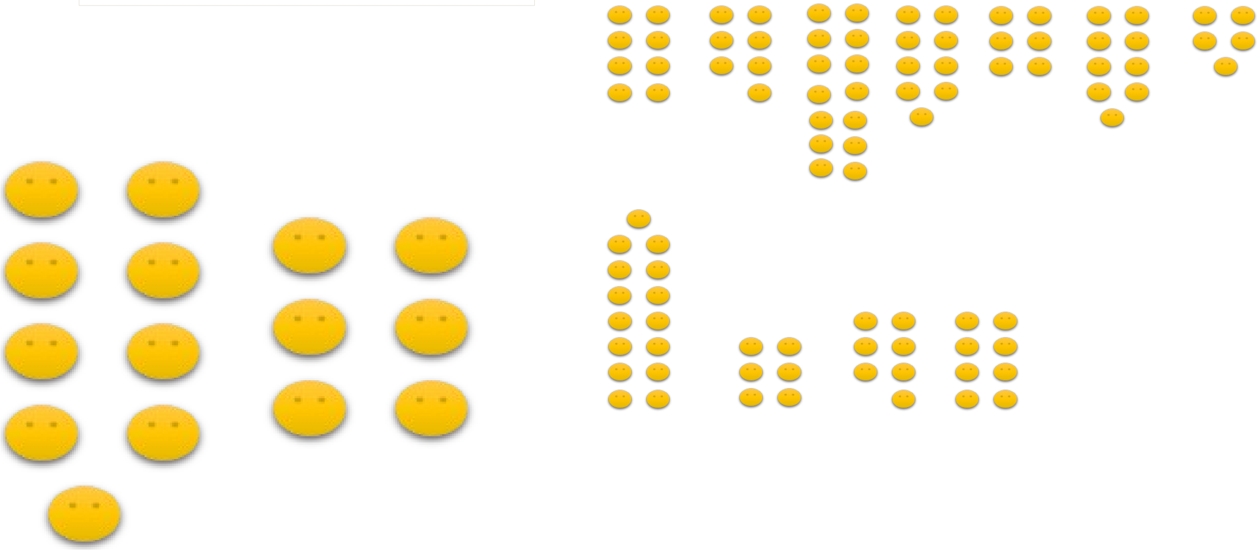
Bin packing

First fit method

5. group has
6 persons:
put persons to bus 2, too



BUS 2

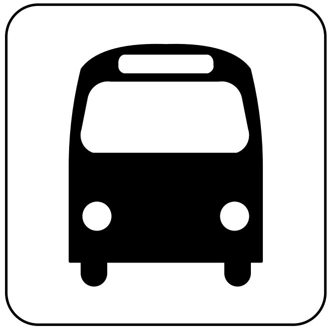


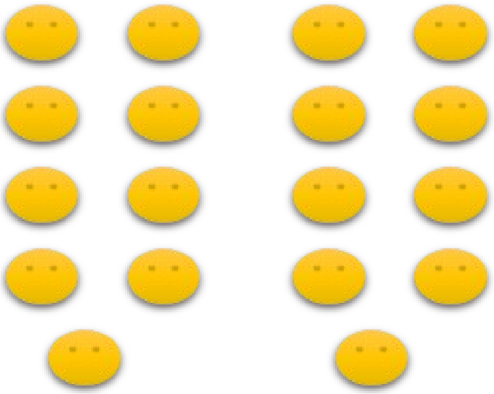
Code School

Bin packing

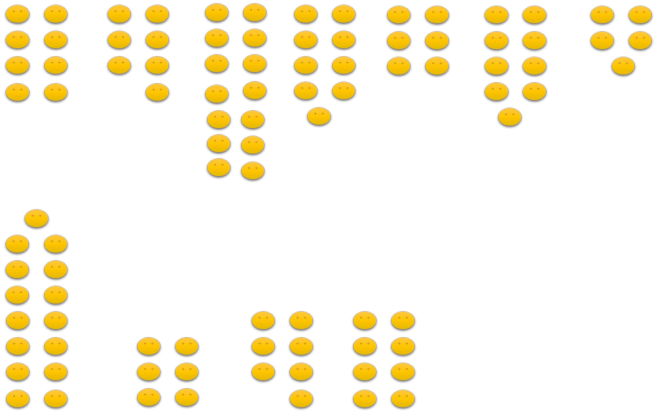
First fit method

6. group has
9 persons:
put persons to bus 3, too





BUS 3

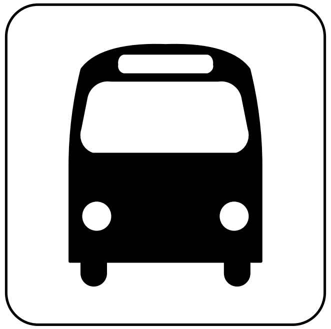


Code School

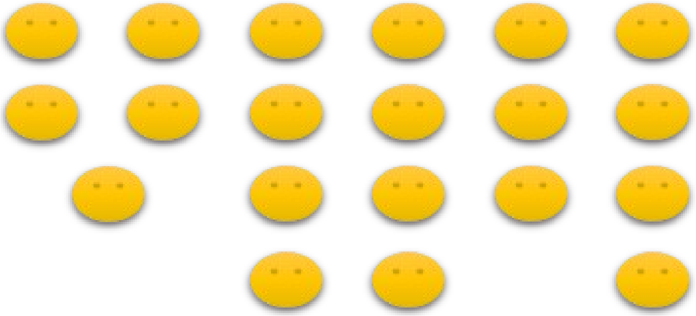
Bin packing

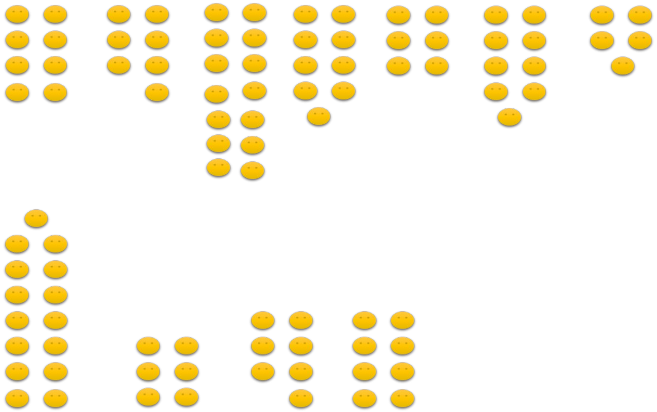
First fit method

7. group has
5 persons:
put persons to bus 1, too



BUS 1



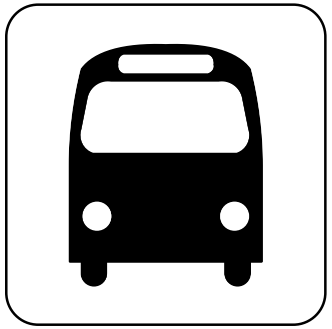


Code School

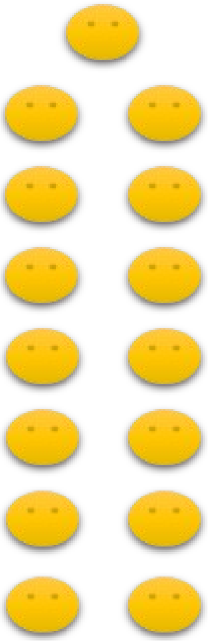
Bin packing

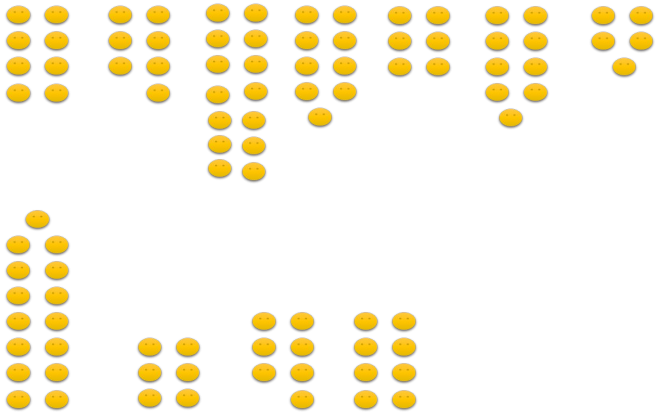
First fit method

8. group has
15 persons:
put persons to bus 4



BUS 4

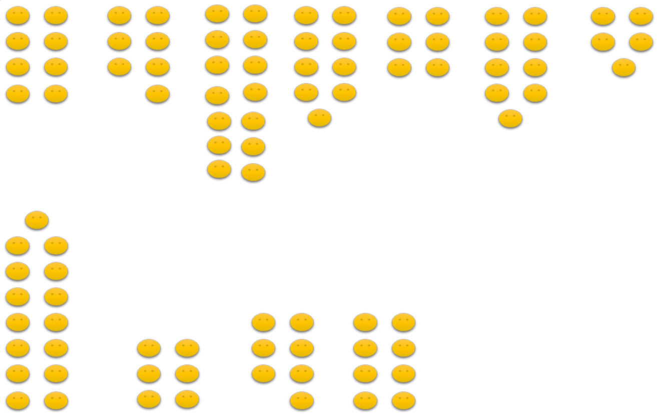




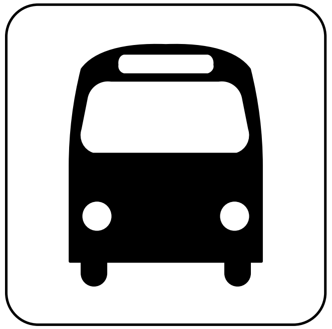
Code School

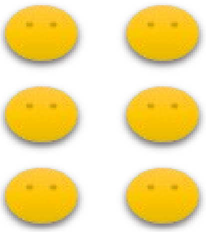
Bin packing

First fit method



9. group has
6 persons:
put persons to bus 5



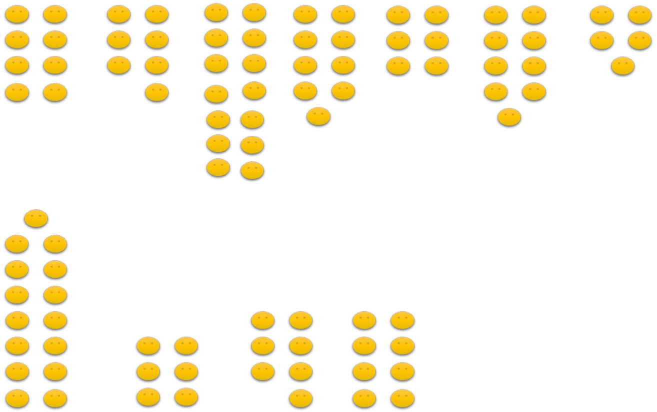


BUS 5

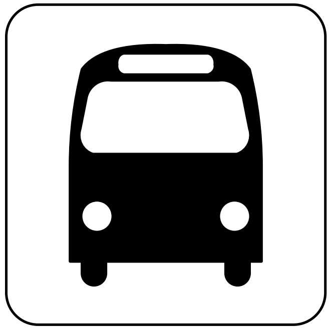
Code School

Bin packing

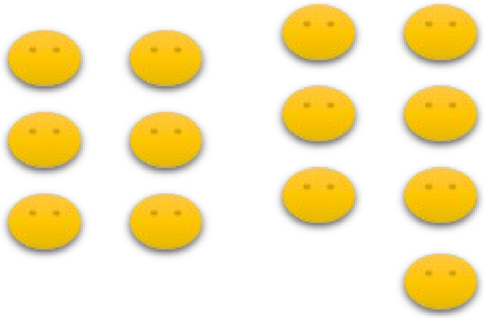
First fit method



10. group has
7 persons:
put persons to bus 5, too



BUS 5

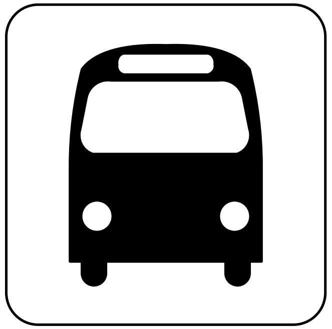


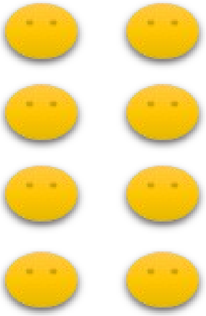
Code School

Bin packing

First fit method

11. group has
8 persons:
put persons to bus 6





BUS 6

Code School

Pascal Triangle

```
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5   10  10  5   1
        1   6   15  20  15  6   1
      1   7   21  35  35  21  7   1
    1   8   28  56  70  56  28  8   1
  1   9   36  84  126 126 84  36  9   1
1   10  45  120 210 252 210 120 45  10  1
```

https://en.wikipedia.org/wiki/Pascal%27s_triangle

Code School

```
                                        1
                                    1       1
                                1       2       1
                            1       3       3       1
                        1       4       6       4       1
                    1       5       10      10      5       1
                1       6       15      20      15      6       1
            1       7       21      35      35      21      7       1
        1       8       28      56      70      56      28      8       1
    1       9       36      84      126     126     84      36      9       1
1       10      45      120     210     252     210     120     45      10      1
```

https://en.wikipedia.org/wiki/Pascal%27s_triangle

Code School

We add first coefficients
to an array – first we create an
array that contains zeroes:

```
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5   10  10  5   1
        1   6   15  20  15  6   1
      1   7   21  35  35  21  7   1
    1   8   28  56  70  56  28  8   1
  1   9   36  84  126 126 84  36  9   1
1   10  45  120 210 252 210 120 45  10  1
```

```
int max = 11;
int r, c;
int base[11][60];
for (r = 0; r < 11; r++)
 for (c = 0; c < 60; c++)
    base[r][c] = 0;
```

Code School

We add first coefficients to an array…
We add there the first 1

```
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5   10  10  5   1
        1   6   15  20  15  6   1
      1   7   21  35  35  21  7   1
    1   8   28  56  70  56  28  8   1
  1   9   36  84  126 126 84  36  9   1
1   10  45  120 210 252 210 120 45  10  1
```

```
base[0][30] = 1;
```

Code School

We add first coefficients to an array…

```
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5  10  10   5   1
        1   6  15  20  15   6   1
      1   7  21  35  35  21   7   1
    1   8  28  56  70  56  28   8   1
  1   9  36  84 126 126  84  36   9   1
1  10  45 120 210 252 210 120  45  10   1
```

```
for (r = 1; r < 11; r++)
{
    for (c = 1; c < 59; c++)
    {
        base[r][c] =  base[r-1][c-1] +  base[r-1][c+1];
    }
}
```

Code School

```
                    1
                 1     1
              1     2     1
           1     3     3     1
        1     4     6     4     1
     1     5     10    10    5     1
  1     6     15    20    15    6     1
1     7     21    35    35    21    7     1
1     8     28    56    70    56    28    8     1
1     9     36    84    126   126   84    36    9     1
1     10    45    120   210   252   210   120   45    10    1
```

Print first with zeroes

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 2 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 3 0 3 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 4 0 6 0 4 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 5 0 10 0 10 0 5 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 6 0 15 0 20 0 15 0 6 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 7 0 21 0 35 0 35 0 21 0 7 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 8 0 28 0 56 0 70 0 56 0 28 0 8 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 9 0 36 0 84 0 126 0 126 0 84 0 36 0 9 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 10 0 45 0 120 0 210 0 252 0 210 0 120 0 45 0 10 0 1 0 0 0 0 0 0 0 0 0 0 0 0
```

Code School

```
                              1
                           1     1
                        1     2     1
                     1     3     3     1
                  1     4     6     4     1
               1     5    10    10     5     1
            1     6    15    20    15     6     1
         1     7    21    35    35    21     7     1
      1     8    28    56    70    56    28     8     1
   1     9    36    84   126   126    84    36     9     1
1    10    45   120   210   252   210   120    45    10     1
```

Adjust printing:

```
for (r = 0; r < 11; r++)
{
    for (c = 0; c < 60; c++)
     if (base[r][c] == 0)
         printf("  ");
     else
         printf("%3d",base[r][c]);

    printf("\n");
}
```

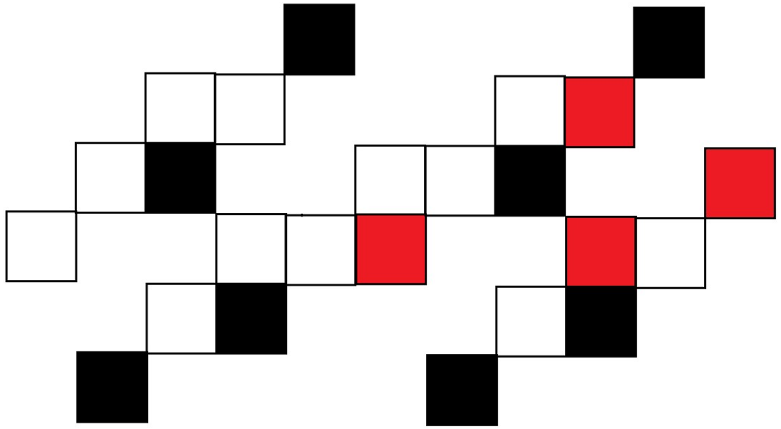
Code School

Adjust printing:

```
for (r = 0; r < 11; r++)
{
    for (c = 0; c < 60; c++)
     if (base[r][c] == 0)
         printf("  ");
     else
         printf("%3d",base[r][c]);

    printf("\n");
}
```

```
                          1
                       1     1
                    1     2     1
                 1     3     3     1
              1     4     6     4     1
           1     5    10    10     5     1
        1     6    15    20    15     6     1
     1     7    21    35    35    21     7     1
  1     8    28    56    70    56    28     8     1
1     9    36    84   126   126    84    36     9     1
1    10    45   120   210   252   210   120    45    10     1
```

Recursive functions



Kakelino’s Code School

Recursive functions

Functions that call themselves.

Function instances are created to RAM memory (stack)

There has to be a condition that stops running.

When all runs have been done, all function instances are deconstructed.

Kakelino's Code School

Recursive functions

Factorial

Factorial(0) is 1

Factorial(1) is 1

Factorial(n) = n * Factorial(n-1)

Kakelino's Code School

Recursive functions

Factorial

Factorial(0) is 1
Factorial(1) is 1
Factorial(n) = n * Factorial(n-1)

```
int factorial(int n)
{
if (n == 0 || n == 1)
  return 1;
else
 return n * factorial(n-1);
}
```

Kakelino's Code School

Recursive functions

```
int factorial(int n)
{
if (n == 0 || n == 1)
  return 1;
else
 return n * factorial(n-1);
}
```

Simulation (what function instances are created)

n is now 4

function call is factorial(4)

1. run: 4 * factorial(3)
2. run: 3 * factorial(2)
3. run: 2 * factorial(1)
4. run: 1 * factorial(0)

Deconstruction:

from run 4 we get 1*1 = 1

from run 3 we get 2*1 = 2

from run 2 we get 3*2 = 6

from run 1 we get 4*6 = **24**

Kakelino's Code School

Recursive functions

Fibonacci

index	1	2	3	4	5	6
value	1	1	2	3	5	8

Kakelino's Code School

Recursive functions

Fibonacci

index	1	2	3	4	5	6
value	1	1	2	3	5	8

```
int fibo(int n)
    {
        if (n == 1 || n == 2)
          return 1;
        else
        {
          return (fibo(n-1) + fibo(n-2));
        }
    }
```

Kakelino's Code School

Fibonacci

index	1	2	3	4	5	6
value	1	1	2	3	5	8

Recursive functions

```
int fibo(int n)
    {
    static int sum = 0;
      if (n == 1 || n == 2)
       sum = 1;
      else
      {
        printf("n is now %d ", n);
        printf("n-1 is now %d  ", n-1);
        printf("n-2 is now %d ", n-2);
        printf("sum is now %d  \n", sum);
        sum = (fibo(n-1) + fibo(n-2));
      }

      return sum;
    }
```

```
n is now 5 n-1 is now 4  n-2 is now 3 sum is now 0
n is now 4 n-1 is now 3  n-2 is now 2 sum is now 0
n is now 3 n-1 is now 2  n-2 is now 1 sum is now 0
n is now 3 n-1 is now 2  n-2 is now 1 sum is now 3

Fibo is  on 5
```

To illustrate simulation some additions!

Variable sum (Fibonacci) is incremented twice after last print…

Recursive functions

```
GCD  is  on 9
```

```c
//greatest common divisor
int gcd(int a, int b)
{
    if (b) return gcd(b , a % b);
        else return a;
}
```

```c
int res = gcd(27,18);
printf("\nGCD  is  on %d \n",res);
```

Kakelino's Code School

Recursive functions

```
sum is   on 15
```

```
// sum of integer values n .. 1
int sum(int val)
{
    if (!val) return val;     /* returns 0 */
    else return val + sum(val-1);
}
```

```
int res = sum(5);
printf("\nsum is  on %d \n",res);
```

Code School

Statistics

Combinations formula is

n = whole population
k = sample

$$n!/k!(n-k)!$$

Code School

Statistics

$n!/k!(n-k)!$

Combinations formula is

n = whole population
k = sample

Example
we have 4 students
how many different pairs can we form
n = 4
k = 2
n! = 1*2*3*4 = 24
k! = 1*2 = 2
(n-k)! = (4-2)! = 2! = 2
Combinations = 24/2*2 = **6**

Code School

Statistics

n!/k!(n-k)!

Example
we have 4 students
how many different pairs can we form
n = 4
k = 2
n! = 1*2*3*4 = 24
k! = 1*2 = 2
(n-k)! = (4-2)! = 2! = 2
Combinations = 24/2*2 = **6**

What are those combinations?
If students are A,B,C and D,
We get
A B
A C
A D
B C
B D
C D
6 possible pairs!

Code School

Statistics

n!/k!(n-k)!

```
static long factorial(int v)
{
    long f = 1;
    for (int i = 1; i <= v; i++)
        f *= i;

    return f;
}

static int combin(int n, int k)
{
    int c = (int) (factorial(n)/factorial(k) * factorial(n-k));
    return c;
}
```

Code School

Statistics

n!/k!(n-k)!

```java
static long factorial(int v)
{
    long f = 1;
    for (int i = 1; i <= v; i++)
        f *= i;

    return f;
}

static int combin(int n, int k)
{
    int c = (int) (factorial(n)/factorial(k) * factorial(n-k));
    return c;
}
```

Test run:

```java
System.out.print("Amount of combinations is " + combin(4,2));
```

```
Amount of combinations is 6
```

Code School

Statistics

Linear regression line

$$y = ax + b$$

Factors a and b can be calculated like this:

$$b = (n\Sigma x_i y_i - \Sigma x_i \Sigma y_i)/(n(\Sigma x_i^2 - (\Sigma x_i)^2)$$

$$a = y\_ - bx\_$$

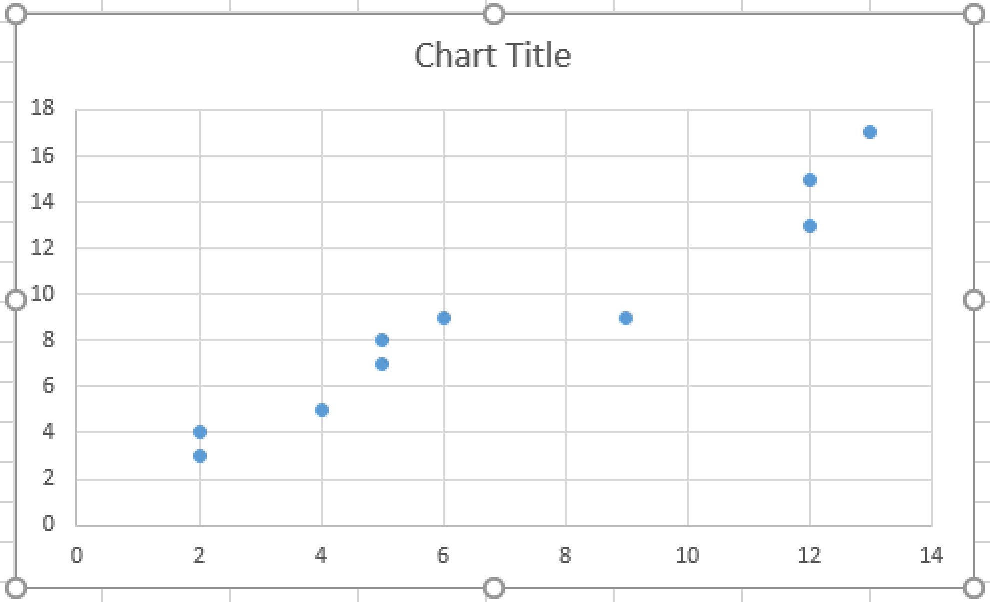
y_ and x_ are averages of x and y

Code School

Statistics

Regression

2	3
2	4
4	5
5	7
5	8
6	9
9	9
12	13
12	15
13	17



Excel gives this result

Code School

Statistics

Regression

Java code:

```
static double[] regr(double[][] points)
{
  double s1 = 0, s2 = 0, s3 = 0, s4 = 0, n = 10;

  for (int k = 0; k < 10; k++)
  {
    s1 = s1 + points[k][0] *  points[k][1];
    s2 = s2 + points[k][0];
    s3 = s3 + points[k][1];
    s4 = s4 + points[k][0] * points[k][0];
  }

  double b = (10*s1 - s2 * s3) / (n* s4 - s2*s2);
  double a = s3/10 - b * s2/10;

  double[] ab = {a,b};

  return ab;

}
```

Code School

Statistics

Regression

Java code:

```
double points[][] = { {2,3}, {2,4}, {4,5}, {5,7}, {5,8}, {6,9}, {9,9}, {12,13}, {12,15}, {13,17} };

double[] factors = regr(points);

System.out.println("Factor a  is " + factors[0]);
System.out.println("Factor b  is " + factors[1]);
```

Code School

Statistics

Regression

Java code:

```
double points[][] = { {2,3}, {2,4}, {4,5}, {5,7}, {5,8}, {6,9}, {9,9}, {12,13}, {12,15}, {13,17} };

double[] factors = regr(points);

System.out.println("Factor a  is " + factors[0]);
System.out.println("Factor b  is " + factors[1]);
```

Code gives these results

```
Factor a  is 1.4240506329113929
Factor b  is 1.0822784810126582
```

Code School

Statistics

Regression

Code gives these results

```
Factor a  is 1.4240506329113929
Factor b  is 1.0822784810126582
```

We use values in Excel

Factor a is 1.4240506329113929			
Factor b is 1.0822784810126582			
2	3,93038		
3	5,35443		
4	6,778481		
5	8,202532		
6	9,626582		
7	11,05063		
8	12,47468		
9	13,89873		
10	15,32278		
11	16,74684		

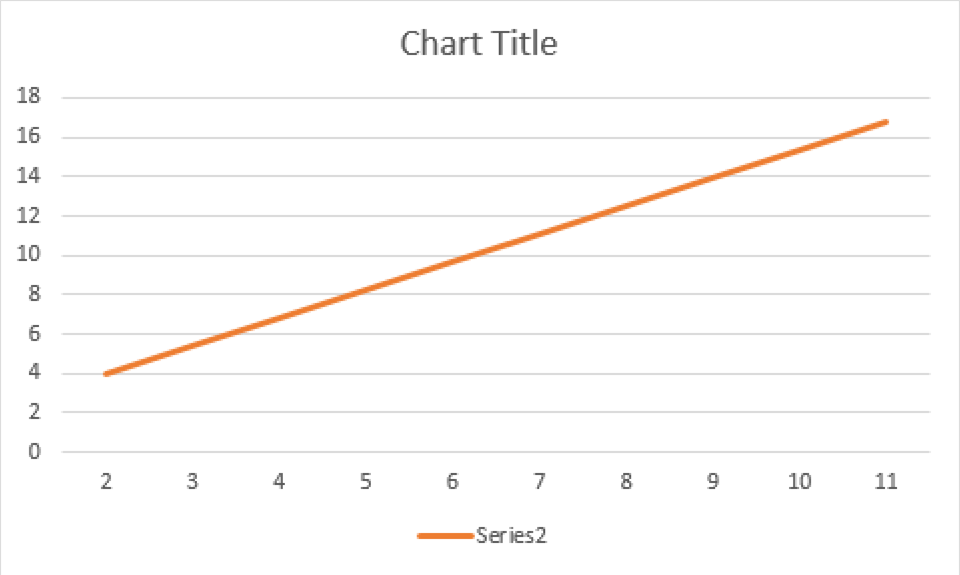
Code School

Statistics

Regression

We use values in Excel

Factor a is 1.4240506329113929	
Factor b is 1.0822784810126582	
2	3,93038
3	5,35443
4	6,778481
5	8,202532
6	9,626582
7	11,05063
8	12,47468
9	13,89873
10	15,32278
11	16,74684



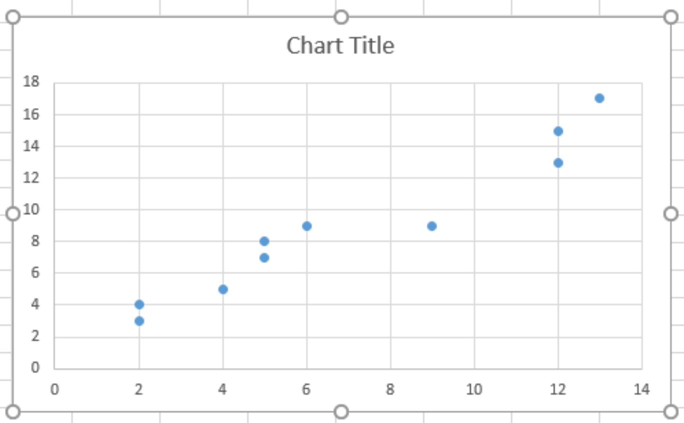
Regression line looks like this

Code School

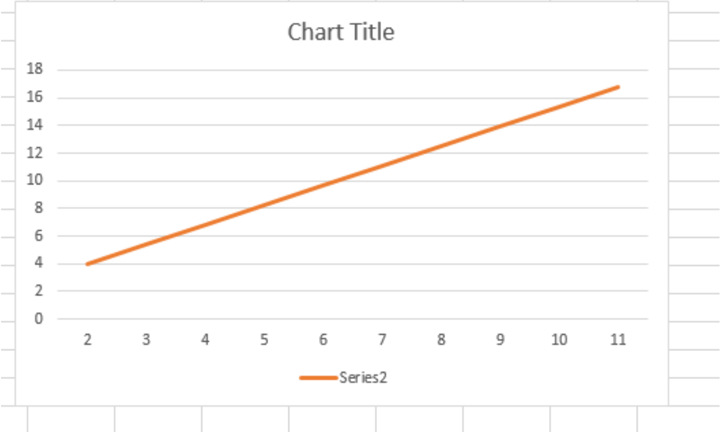
Statistics

Regression

2	3
2	4
4	5
5	7
5	8
6	9
9	9
12	13
12	15
13	17



Here we have points and the line

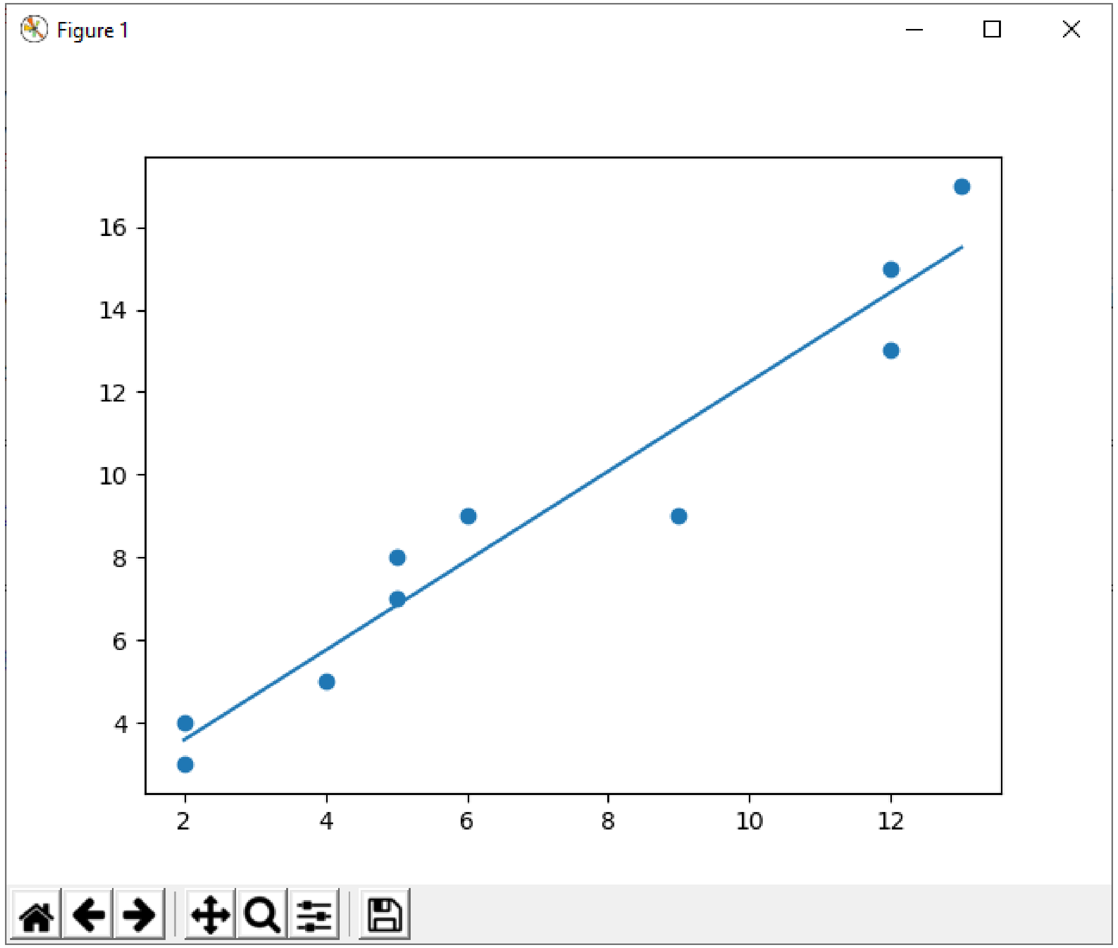


Code School

Statistics

Regression

Here points and line are shown by Python



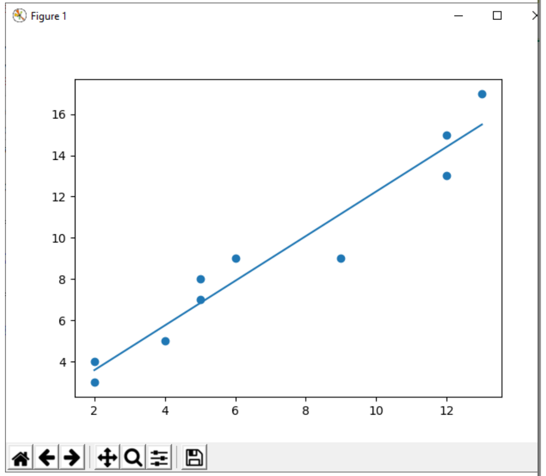
Code School

Statistics

Regression

Here points and line are shown by Python

Code



regress1.py - C:/kk/2018-2019/PYTHON/regress1.py (3.7.1)

File Edit Format Run Options Window Help

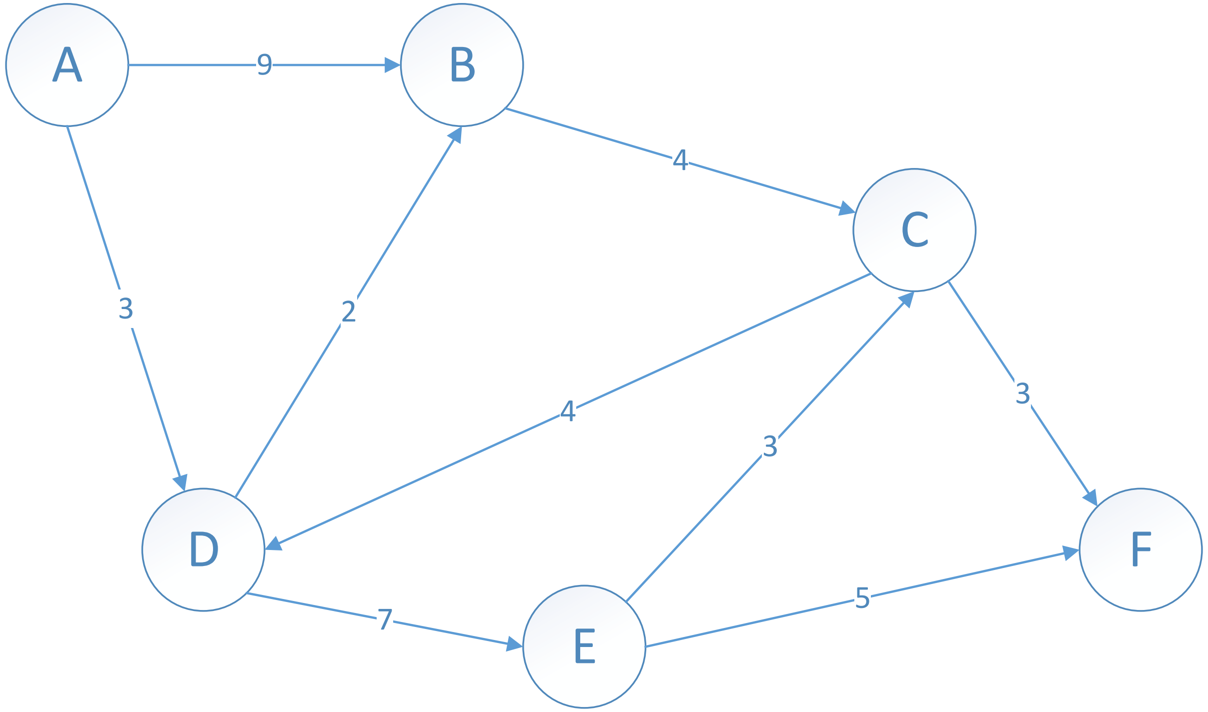
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt   #graphics libs
import seaborn as sns
import matplotlib as mpl
# {2,3}, {2,4}, {4,5}, {5,7}, {5,8}, {6,9}, {9,9}, {12,13}, {12,15}, {13,17}

X = [2,2,4,5,5,6,9,12,12,13]
Y = [3,4,5,7,8,9,9,13,15,17]
# solve a and b  (line)
def best_fit(X, Y):
   xbar = sum(X)/len(X)
   ybar = sum(Y)/len(Y)
   n = len(X) # or len(Y)

   numer = sum([xi*yi for xi,yi in zip(X, Y)]) - n * xbar * ybar
   denum = sum([xi**2 for xi in X]) - n * xbar**2

   b = numer / denum
   a = ybar - b * xbar
   print('best fit line:\ny = {:.2f} + {:.2f}x'.format(a, b))
   return a, b
# regr.line
a, b = best_fit(X, Y)
# plotting
import matplotlib.pyplot as plt
plt.scatter(X, Y)
yfit = [a + b * xi for xi in X]
plt.plot(X, yfit)
plt.show()
```

Ln: 38   Col: 0

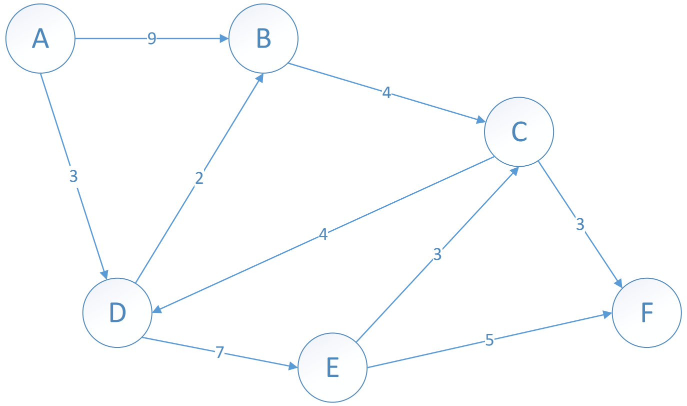
Ford-Fulkerson

Simulation

Ford-Fulkerson Simulation



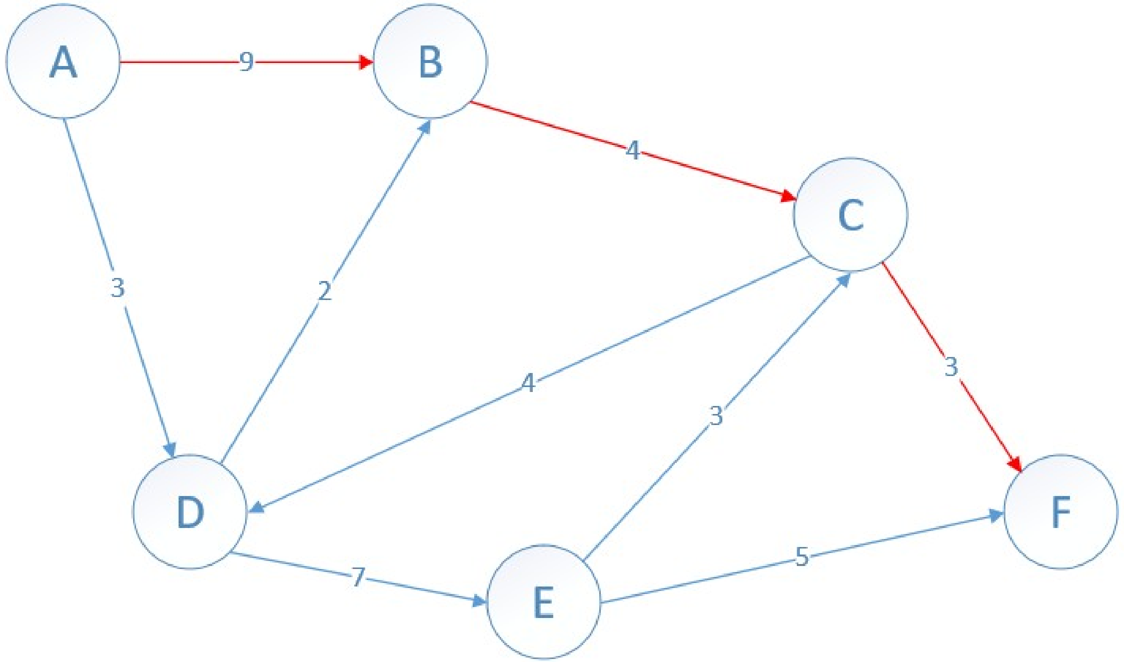
Ford-Fulkerson Simulation



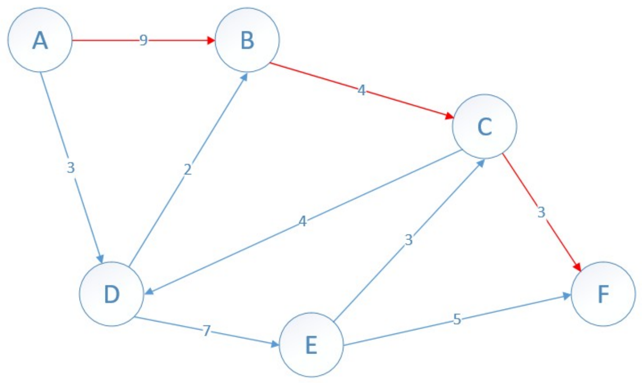
Let's create a table that helps us in book keeping

Arc (Route)	Minimun capacity	Remaining capacity

Ford-Fulkerson Simulation

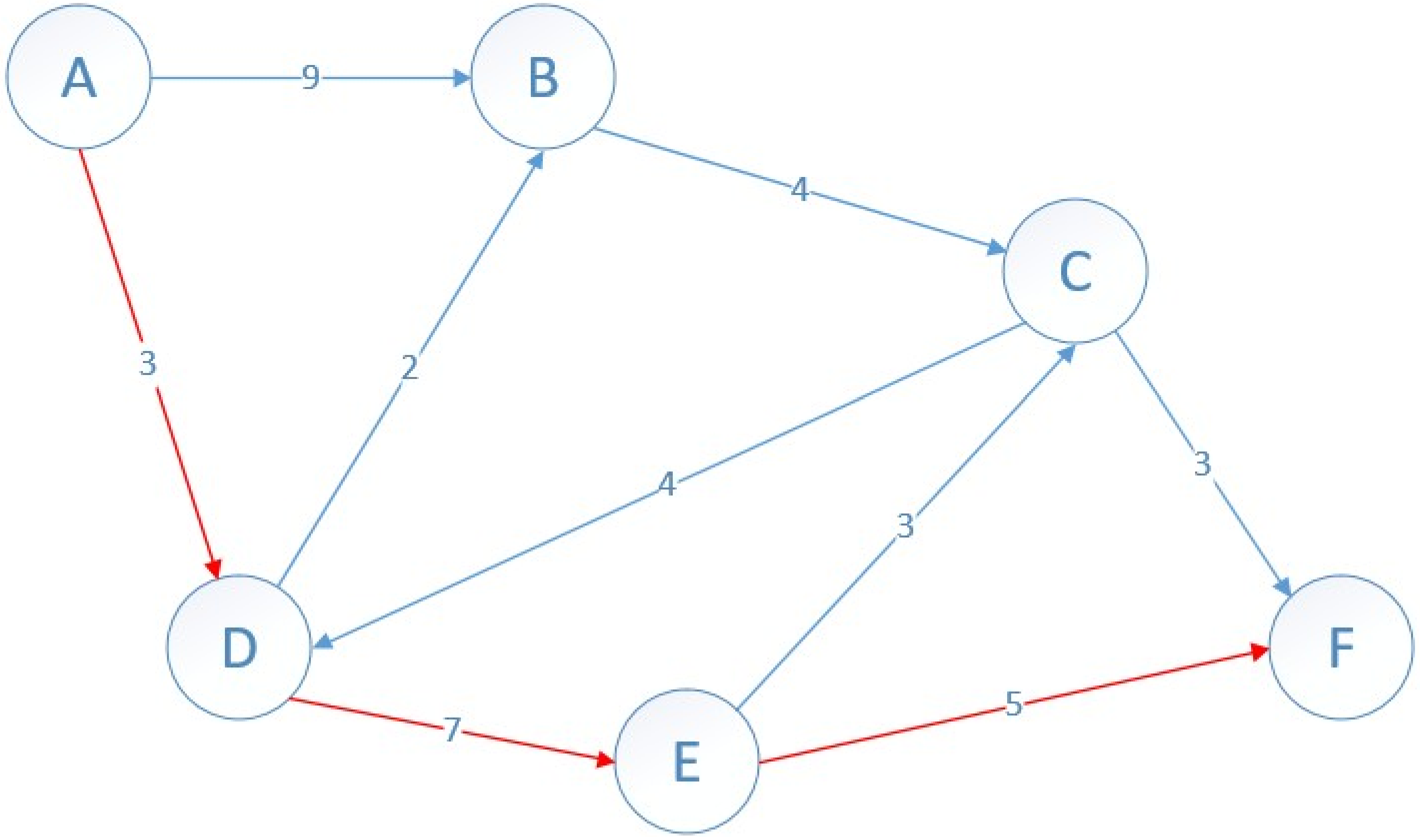


Ford-Fulkerson Simulation

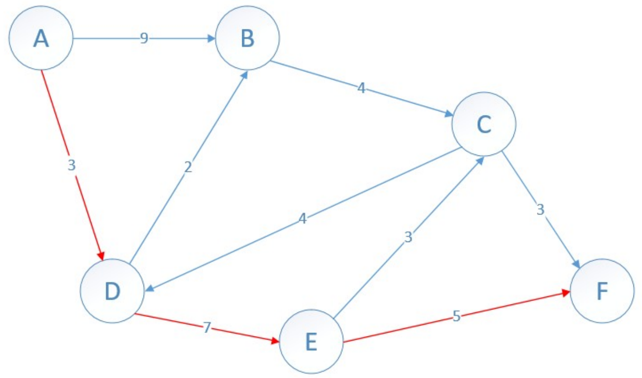


Arc (Route)	Minimun capacsity	Remaining capacity
A-B-C-D	3	A-B: 9–6=3 B-C: 4-4=1 C-D: 3-3=0

Ford-Fulkerson
Simu

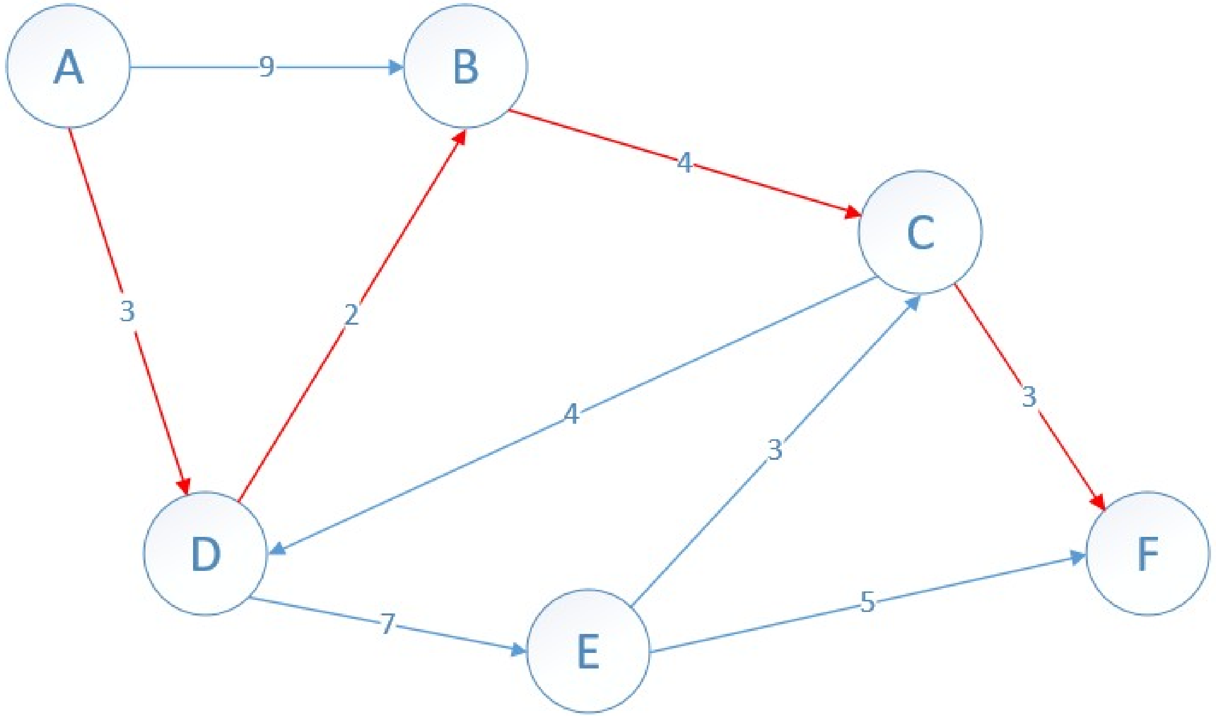


Ford-Fulkerson Simulation

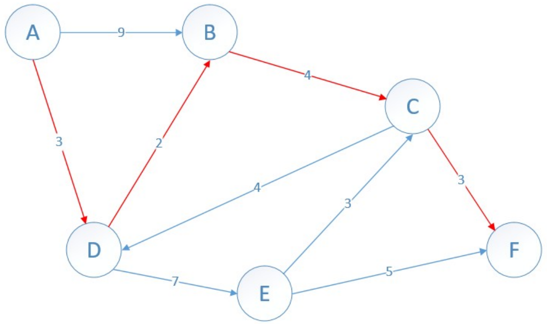


Arc (Route)	Minimun capacity	Remaining capacity
A-B-C-D	3	A-B: 9–6=3 B-C: 4-4=1 C-D: 3-3=0
A-D-E-F	3	A-D:3-3=0 D-E:7-3=4 E-F:5-3=2

Ford-Fulkerson Simulation

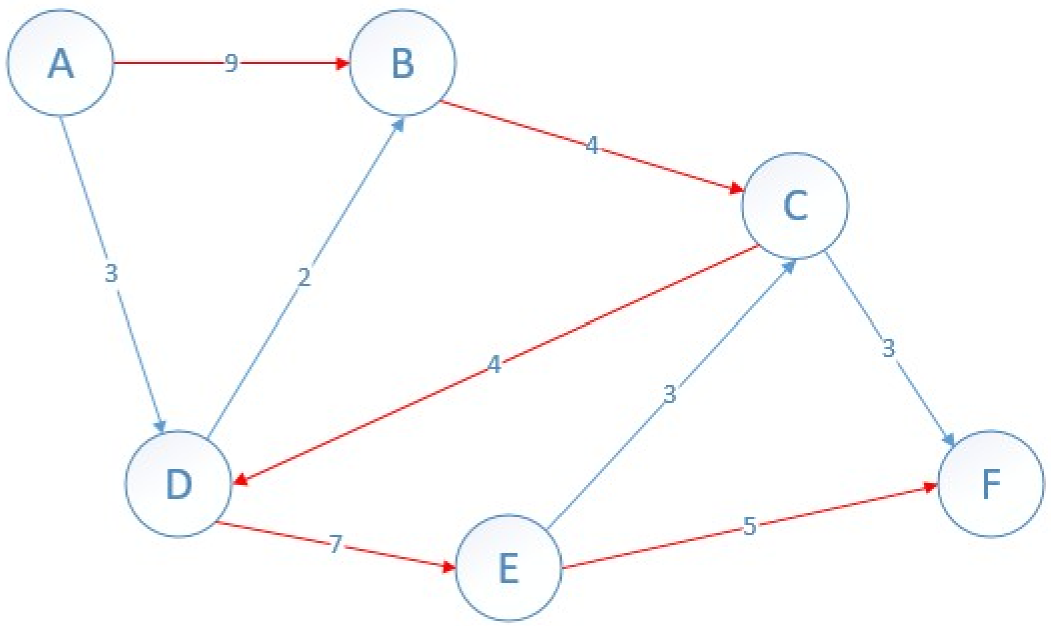


Ford-Fulkerson Simulation

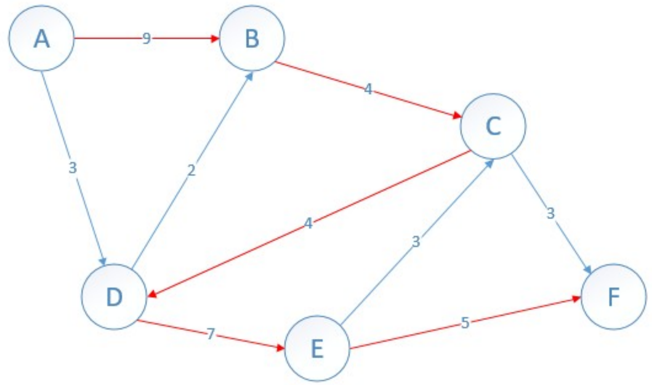


Arc (Route)	Minimun capacsity	Remaining capacity
A-B-C-D	3	A-B: 9–6=3 B-C: 4-4=1 C-D: 3-3=0
A-D-E-F	3	A-D:3-3=0 D-E:7-3=4 E-F:5-3=2
A-D-B-C-F	0	Nothing

Ford-Fulkerson Simulation

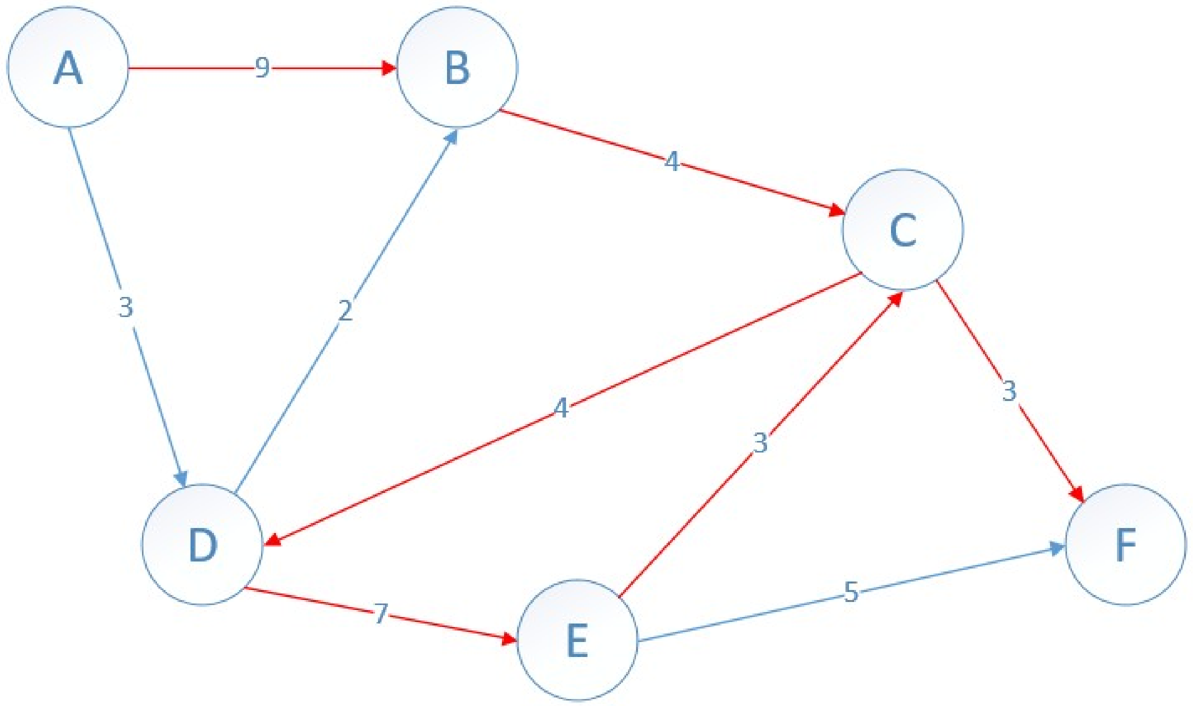


Ford-Fulkerson Simulation

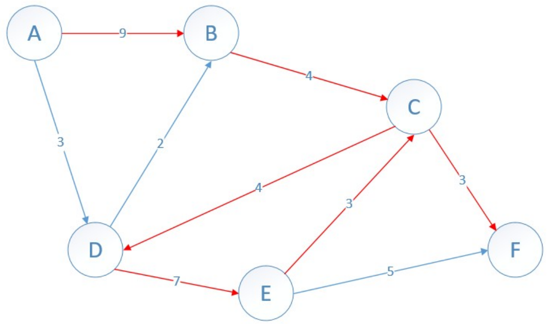


Arc (Route)	Minimun capacity	Remaining capacity
A-B-C-D	3	A-B: 9–6=3 B-C: 4-4=1 C-D: 3-3=0
A-D-E-F	3	A-D:3-3=0 D-E:7-3=4 E-F:5-3=2
A-D-B-C-F	0	Nothing
A-B-C-D-E-F	1	A-B:3-1=2 B-C:1-1=0 C-D:4-1=3 D-E:4-1=3 E-F:2-1=1

Ford-Fulkerson Simulation



Ford-Fulkerson Simulation

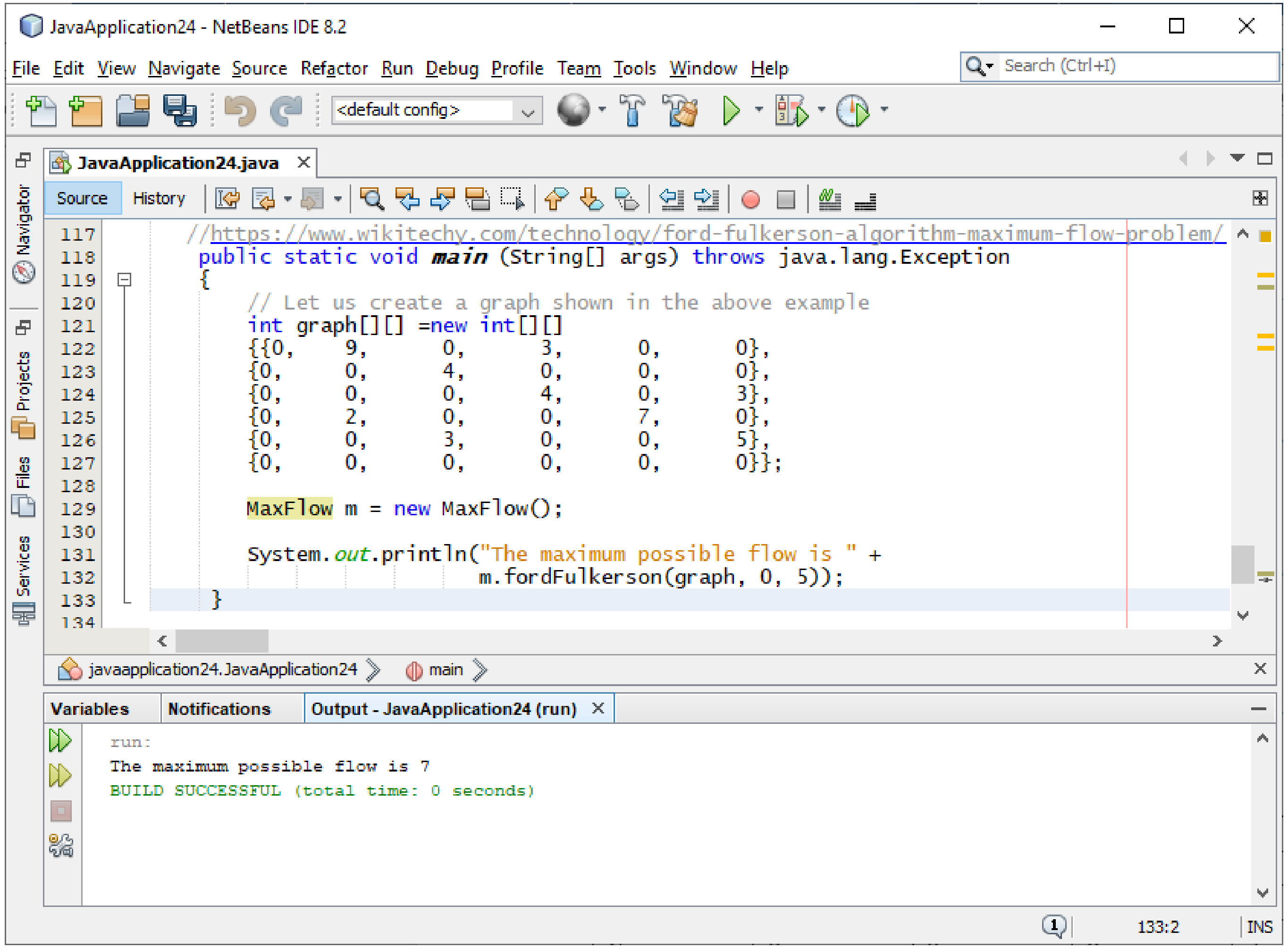


Arc (Route)	Minimun capacity	Remaining capacity
A-B-C-D	3	A-B: 9–6=3 B-C: 4-4=1 C-D: 3-3=0
A-D-E-F	3	A-D:3-3=0 D-E:7-3=4 E-F:5-3=2
A-D-B-C-F	0	Nothing
A-B-C-D-E-F	1	A-B:3-1=2 B-C:1-1=0 C-D:4-1=3 D-E:4-1=3 E-F:2-1=1
A-B-C-D-E-C-F	0	Nothing
MAX Capasity	7	

Ford-Fulkerson Simulation

	A	B	C	D	E	F
A	0	9	0	3	0	0
B	0	0	4	0	0	0
C	0	0	0	4	0	3
D	0	2	0	0	7	0
E	0	0	3	0	0	5
F	0	0	0	0	0	0

Ford-Fulkerson Simulation



Ford-Fulkerson Simulation

DSP & FFT

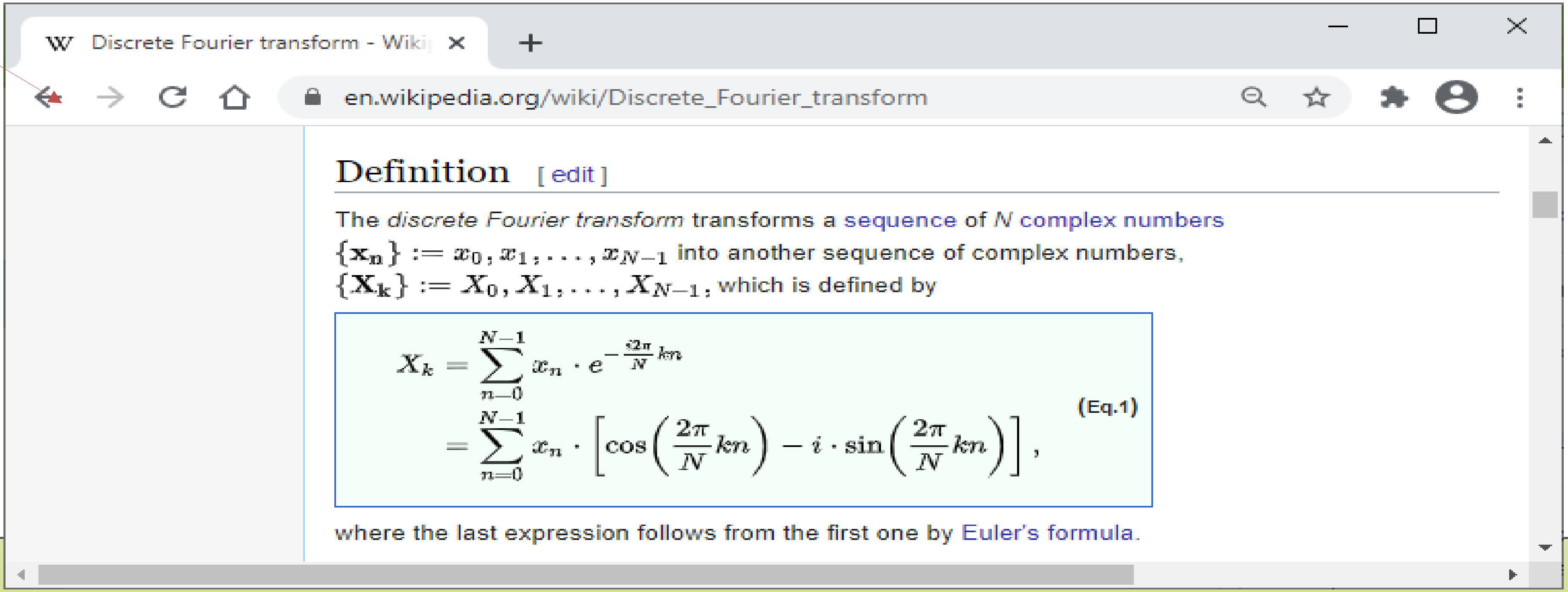
Main sources
https://www.dspguide.com
Chapter 12 is excellent here

There are several ways to calculate the Discrete Fourier Transform (DFT), such as solving simultaneous linear equations or the *correlation* method described in Chapter 8. The Fast Fourier Transform (FFT) is another method for calculating the DFT. While it produces the same result as the other approaches, it is incredibly more efficient, often reducing the computation time by *hundreds*. This is the same improvement as flying in a jet aircraft versus walking! If the FFT were not available, many of the techniques described in this book would not be practical. While the FFT only requires a few dozen lines of code, it is one of the most complicated algorithms in DSP. But don't despair! You can easily use published FFT routines without fully understanding the internal workings.

Kakelino’s Code School

DSP & FFT

Main sources
https://www.dspguide.com
Chapter 12 is excellent here
Wikipedia



DFT

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}$$

$$x(n) = 1/N \sum_{k=0}^{N-1} X(k) W_N^{-kn}$$

$$W_N = e^{-j2\pi/N}$$

DFT

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}$$

$$x(n) = 1/N \sum_{k=0}^{N-1} X(k) W_N^{-kn}$$

$$W_N = e^{-j2\pi/N}$$

Example

$X(k) \rightarrow x(n)$

DFT

Example

$X(k) \rightarrow x(n)$

X(k) can be now 1, 3/4, 1/2, 1/4

$X(0) = 1$
$X(1) = 3/4$
$X(2) = 1/2$
$X(3) = 1/4$

Transformation formula is

$$x(n) = 1/N \sum_{k=0}^{N-1} X(k) W_N^{-kn}$$

where

$W_N = e^{-j2\pi/N}$

Example

$X(k) \rightarrow x(n)$

X(k) can be now 1, 3/4, 1/2, 1/4

$X(0) = 1$
$X(1) = 3/4$
$X(2) = 1/2$
$X(3) = 1/4$

Transformation formula is

$$x(n) = 1/N \sum_{k=0}^{N-1} X(k) W_N^{-kn}$$

where

$$W_N = e^{-j2\pi/N}$$

When N = 4 we get

$$x(n) = 1/4 \sum_{k=0}^{3} X(k)\, e^{j2\pi nk/4}$$

DFT

Example

$X(k) \to x(n)$

X(k) can be now 1, 3/4, 1/2, 1/4

$X(0) = 1$
$X(1) = 3/4$
$X(2) = 1/2$
$X(3) = 1/4$

Transformation formula is

$$x(n) = 1/N \sum_{k=0}^{N-1} X(k) W_N^{-kn}$$

where

$$W_N = e^{-j2\pi/N}$$

When N = 4 we get

$$x(n) = 1/4 \sum_{k=0}^{3} X(k)\, e^{j2\pi nk/4}$$

Value that is corresponding fo X(0) can be calculated like this

$$1/4(\, X(0)\, e^{j2\pi n0/4}\,) = 1/4\,(\,1 * e^{0}\,) = 1/4 * 1*1 = 1/4\,(1)$$

DFT

Example

X(k) -> x(n)

X(k) can be now 1, 3/4, 1/2, 1/4

$X(0) = 1$
$X(1) = 3/4$
$X(2) = 1/2$
$X(3) = 1/4$

Transformation formula is

$$x(n) = 1/N \sum_{k=0}^{N-1} X(k) W_N^{-kn}$$

where

$$W_N = e^{-j2\pi/N}$$

Value that is corresponding fo X(0) can be calculated like this

$$1/4(X(0)\, e^{j2\pi n0/4}) = 1/4 (1 * e^0) = 1/4 * 1*1 = 1/4 \ (1)$$

When N = 4 we get

$$x(n) = 1/4 \sum_{k=0}^{3} X(k)\, e^{j2\pi nk/4}$$

Other values can be calculated in the same way

$$x(n) = 1/4\{1 + X(1)\, e^{j2\pi n1/4} + X(2)\, e^{j2\pi n2/4} + X(3)\, e^{j2\pi n3/4} \}$$

DFT

Example

X(k) can be now 1, 3/4, 1/2, 1/4

Value that is corresponding fo X(0) can be calculated like this

$$1/4(X(0)\, e^{j2\pi n0/4}) = 1/4\ (1 * e^{0}) = 1/4 * 1*1 = 1/4\ (1)$$

Other values can be calculated in the same way

$$x(n) = 1/4\{1 + X(1)\, e^{j2\pi n1/4} + X(2)\, e^{j2\pi n2/4} + X(3)\, e^{j2\pi n3/4}\}$$

Common formula is then

$$\mathbf{x(n) = 1/4\{1 + 3/4\, e^{j2\pi n1/4} + 1/2\, e^{j2\pi n2/4} + 1/4\, e^{j2\pi n3/4}\}}$$

As an example x(0) is calculated:

$$x(0) = 1/4\{1 + 3/4\, e^{j2\pi 01/4} + 1/2\, e^{j2\pi 02/4} + 1/4\, e^{j2\pi 03/4}\}$$

$$= 1/4\ (1+3/4+1/2+1/4) = 5/8$$

DFT

Example

X(k) can be now 1, 3/4, 1/2, 1/4

Common formula is then

$$\mathbf{x(n) = 1/4\{1 + 3/4\ e^{j2\pi n1/4} + 1/2\ e^{j2\pi n2/4} + 1/4\ e^{j2\pi n3/4}\}}$$

As an example x(0) is calculated:

$$x(0) = 1/4\{1 + 3/4\ e^{j2\pi 01/4} + 1/2\ e^{j2\pi 02/4} + 1/4\ e^{j2\pi 03/4}\}$$

$$= 1/4\ (1+3/4+1/2+1/4) = 5/8$$

DFT

Example

X(k) can be now 1, 3/4, 1/2, 1/4

Now we can try to manage the task without using complex values.

By using Euler's formula

Imag.unit is only in the exponent

$e^{i\varphi} = \cos\varphi + i\sin\varphi$

we get

$$x(n) = 1/4(1 + 3/4(\cos 2\pi n1/4 + j\sin 2\pi n1/4) + 1/2(\cos 2\pi n2/4 + j\sin 2\pi n2/4) + 1/4(\cos 2\pi n3/4 + j\sin 2\pi n3/4)$$

AND then

$$x(n) = 1/4(1 + 3/4(\cos \pi n1/2 + j\sin \pi n1/2) + 1/2(\cos \pi n + j\sin \pi n) + 1/4(\cos \pi n3/2 + j\sin \pi n3/2)$$

The final formula is

$$x(n) = 1/N \sum_{k=0}^{N-1} [X(k) (\cos 2\pi nk/N + j\sin 2\pi nk/N)]$$

DFT Example X(k) can be now 1, 3/4, 1/2, 1/4

Now we can try to manage the task without using complex values.

By using Euler's formula Imag.unit is only in the exponent

$e^{i\varphi} = \cos\varphi + i\sin\varphi$

we get

$x(n) = 1/4(1 + 3/4(\cos 2\pi n1/4 + j\sin 2\pi n1/4) + 1/2(\cos 2\pi n2/4 + j\sin 2\pi n2/4) + 1/4(\cos 2\pi n3/4 + j\sin 2\pi n3/4)$

AND then

$x(n) = 1/4(1 + 3/4(\cos \pi n1/2 + j\sin \pi n1/2) + 1/2(\cos \pi n + j\sin \pi n) + 1/4(\cos \pi n3/2 + j\sin \pi n3/2)$

The final formula is

$$x(n) = 1/N \sum_{k=0}^{N-1} [X(k) (\cos 2\pi nk/N + j\sin 2\pi nk/N)]$$

C++ code first

```
#include <iostream>
#include <complex>

using namespace std;
#define pi  3.14

int main()
{

    int k, n;
    double X[] = {1, 0.75, 0.5, 0.25, 0, 0.25, 0.5, 0.75};
    int N = 8;
    double x[8];

n =1;
for (n = 0; n < N; n++)
{

    double sum = 0.0;
    for (k=0; k < N; k++)
    {
       sum = sum + (1.0/N * X[k]) * ((cos(2*pi*n*k/N)) + ( real(complex<double>(0,sin(2*pi*n*k/N)) ) ) );
    }
    cout << sum <<  " ";
}
}
```

```
C:\CODES\dsp.exe
0.5 0.213113 -0.000397368 0.0363842 4.12184e-006 0.0369988 0.00120156 0.215307
```

DFT

C# code

```
static void Main(string[] args)
{
    double[] X = {  1, 0.75, 0.5, 0.25, 0, 0.25, 0.5, 0.75 };

    double[] x = new double[8];
    double  N = 8;
    for (int n = 0; n < N; n++)
    {
        double sum = 0;
        for (int k = 0; k < N; k++)
        {
           sum += (1.0 / N * (X[k])) * Math.Cos(2 * Math.PI * n * (double) k / N);
            Complex z = new Complex(0,Math.Sin(2 * Math.PI * n * (double) k / N));
            sum += (1.0 / N * (X[k])) * z.Real;

        }
        Console.Write(" " + sum);
    }
    Console.Read();
}
```

```
0,5 0,213388347648318 -4,01823959847968E-17 0,0366116523516816 0 0,0366116523516815 -1,2925299258627E-16 0,213388347648319
```

DFT

We get faster algorithm by using symmetry and periodicity in formulas (they are called Fast Fourie Transform methods, FFT);

$W_N = e^{-j2\pi/N}$

We get

$W^{k(n-N)}_N = W^{-kn}_N$

and

$W^{kn}_N = W^{k(n+N)}_N = W^{(k+N)n}_N$

DFT using FFT

C++ code first

Part 1

```
#include <iostream>
#include <cmath>
#include <complex>

using namespace std;

complex<double> x[8] = {0.5, 0.21338834764483, 0, 0.036611652, 0, 0.036611652, 0, 0.21338834764483};
// we get {1, 0.75, 0.5, 0.25, 0, 0.25, 0.5, 0.75};
const  double pi = 3.1415926;
int amount();
void fft(int N, complex<double>x[]);
void turn(int N, complex <double>x[]);
```

DFT using FFT
C++ code first

Part 2

```
int main()
{
    int N = 8;
    /*complex temp[8];  bits are turned ...
    temp[0] = x[0];temp[4] = x[1];temp[2] = x[2];temp[6] = x[3];temp[1] = x[4];
    temp[5] = x[5];temp[3] = x[6];temp[7] = x[7];    */

    int e;
    for (e = 0; e < N; e++)
      cout << x[e] << "\n";

    cout << "\n";
    turn(N, x);
    fft(N, x);
    cout << "values" << endl;
    for (e = 0; e < N; e++)
    cout << x[e] << " ";
    cout << "\n";

}
```

DFT using FFT
C++ code first

Part 3

```
void fft(int N, complex<double> x[])
{
        int state = 1, width;
        int S, M, R, order = N;
        complex<double>t1, t2;
        double a;
        for (M = 0; order != 1; M++)
        order = (order >> 1);

    for (int s = 1; s <= M; s++)
    {
        state = pow(2,s);
        S = N/state;
        width = state/2;
        for (int p = 0; p <= (width - 1); p++)
        {
            R = S * p;
            a = 2 * pi * R/N;
            t1 = complex<double>(cos(a), -sin(a));
            for (int o = p; o <= N-2; o = o + state)
            {
                int b = o + width;
                t2 = x[b] * t1;
                x[b] = x[o] - t2;
                x[o] = x[o] + t2;
            }
        }
    }

}
```

DFT using FFT
C++ code first

Part 4

```
void turn(int N, complex<double> x[])
{
    complex <double>temp[8];
    int M, order = N;

    for (M = 0; order != 1; M++)
    order = (order >> 1);

    for (int i = 0; i < N; i++)
    {
    int ind1 = 0;
    int ind2 = i;

    for (int j = 0; j <= M-1; j++)
    {
    ind1 = ind1 + ((( 1 << j ) & ind2) ? (1 << (M-1-j)) : 0
    }

    temp[ind1] = x[i];

    // show bits turning info
    for (int i=0; i < N; i++)
    {
       x[i] = temp[i];
       cout << x[i] << "\n";
    }
}
```

DFT using FFT
C#

Part 1,
FFT function

```
const double pi = Math.PI;
static void fft(int N, Complex[] x)
{
    int state = 1, width;
    int S, M, R, order = N;
    Complex t1, t2;
    double a;
    for (M = 0; order != 1; M++)
        order = (order >> 1);
    for (int s = 1; s <= M; s++)
    {
        state = (int)Math.Pow(2, s);
        S = N / state;
        width = state / 2;
        for (int p = 0; p <= (width - 1); p++)
        {
            R = S * p;
            a = 2 * pi * (double) R / N;
            t1 = new Complex(Math.Cos(a), -Math.Sin(a));

            for (int o = p; o <= N - 2; o = o + state)
            {
                int b = o + width;
                t2 = x[b] * t1;
                x[b] = x[o] - t2;
                x[o] = x[o] + t2;
            }
        }    }      }
```

DFT using FFT
C#

Part 2,
SWAP bits function

```
static void turn(int N, Complex[] x)
{
    Complex[] temp = new Complex[8];
    int M, order = N;
    for (M = 0; order != 1; M++)
        order = (order >> 1);
    for (int i = 0; i < N; i++)
    {
        int ind1 = 0;
        int ind2 = i;

        for (int j = 0; j <= M - 1; j++)
        {
            if (((1 << j) & ind2) != 0)
                ind1 = ind1 + (1 << (M - 1 - j));
            else
                ind1 = ind1 + 0;
        }
        temp[ind1] = x[i];
    }
    // show bits turning info
    for (int i = 0; i < N; i++)
    {
        x[i] = temp[i];
        Console.Write(" " + x[i] + "\n");
    }
}
```

DFT using FFT
C#

Part 3,
Start testing

```
static void testFFT()
{
    Complex[] x = { 0.5, 0.2133883476483, 0, 0.036611652,
                    0, 0.036611652, 0, 0.2133883476483 };
    // we get {1, 0.75, 0.5, 0.25, 0, 0.25, 0.5, 0.75};
    int N = 8;
    /*complex temp[8];  bits are turned ...
    temp[0] = x[0];temp[4] = x[1];temp[2] = x[2];temp[6] = x[3];temp[1] = x[4];
    temp[5] = x[5];temp[3] = x[6];temp[7] = x[7];   */

    int e;
    for (e = 0; e < N; e++)
        Console.Write("" + x[e] + "\n");
    turn(N, x);
    fft(N, x);
    Console.Write("values" + "\n");
    for (e = 0; e < N; e++)
        Console.Write("" + x[e] + "\n");

}
```

DFT using FFT C#

Part 3,
Start testing

```
static void Main(string[] args)
{
    testFFT();
```

```
static void testFFT()
{
    Complex[] x = { 0.5, 0.2133883476483, 0, 0.036611652,
                    0, 0.036611652, 0, 0.2133883476483 };
    // we get {1, 0.75, 0.5, 0.25, 0, 0.25, 0.5, 0.75};
    int N = 8;
    /*complex temp[8];  bits are turned ...
    temp[0] = x[0];temp[4] = x[1];temp[2] = x[2];temp[6] = x[3];temp[1] = x[4];
    temp[5] = x[5];temp[3] = x[6];temp[7] = x[7];   */

    int e;
    for (e = 0; e < N; e++)
        Console.Write("" + x[e] + "\n");
    turn(N, x);
    fft(N, x);
    Console.Write("values" + "\n");
    for (e = 0; e < N; e++)
        Console.Write("" + x[e] + "\n");

}
```

DFT using FFT
C#

TEST run

```
values
(0,9999999992966, 0)
(0,750000000497327, 2,77555756156289E-17)
(0,5, 0)
(0,249999999502673, -2,77555756156289E-17)
(7,03400004908872E-10, 0)
(0,249999999502673, -2,77555756156289E-17)
(0,5, 0)
(0,750000000497327, 2,77555756156289E-17)
```

C++ program results are same

```
values
(1,0) (0.75,6.69872e-009) (0.5,0) (0.25,-1.33974e-008) (7.034e-010,0) (0.25,-6.69872e-009) (0.5,0) (0.75,1.33974e-008)
```

DFT

One idea was to test how C# handles complex values

Another thing was to test
Wikipedias pseudocodes

Third thing was to wonder why FFT is faster than
common Brute force algorithm

Thank You!

Insertion Sort

29	10	14	37	13

Insertion Sort

29	10	14	37	13

Start from the right side
Now there is value 10
Copy the value
Find the right place: if there are bigger ones on the right, move them to the left,
Until you reach the beginning of the array or there is smaller value than 10.
Add value 10 to found new place!

Insertion Sort

29	10	14	37	13

29	10	14	37	13

Copy 10

Start from the right side
Now there is value 10
Copy the value
Find the right place: if there are bigger ones on the right, move them to the left,
Until you reach the beginning of the array or there is smaller value than 10.
Add value 10 to found new place!

Insertion Sort

29	10	14	37	13

29	10	14	37	13

Move 29 to the right

Start from the right side
Now there is value 10
Copy the value
Find the right place: if there are bigger ones on the right, move them to the left,
Until you reach the beginning of the array or there is smaller value than 10.
Add value 10 to found new place!

Insertion Sort

29	10	14	37	13

29	29	14	37	13

Move 29 to the right

Start from the right side
Now there is value 10
Copy the value
Find the right place: if there are bigger ones on the right, move them to the left,
Until you reach the beginning of the array or there is smaller value than 10.
Add value 10 to found new place!

Insertion Sort

29	10	14	37	13

10	29	14	37	13

Add 10 to
the beginning

Start from the right side
Now there is value 10
Copy the value
Find the right place: if there are bigger
ones on the right, move them to the left,
Until you reach the beginning of the array
or there is smaller value than 10.
Add value 10 to found new place!

Insertion Sort

29	10	14	37	13

10	29	14	37	13

Choose next one,
there is 14

Insertion Sort

29	10	14	37	13

10	29	14	37	13

↑

Copy the value

Insertion Sort

29	10	14	37	13

10	29	29	37	13

Move 29 to the right

Insertion Sort

29	10	14	37	13

10	14	29	37	13

Add 14 to its place

Insertion Sort

29	10	14	37	13

10	14	29	37	13

Now, 37

Insertion Sort

29	10	14	37	13

10	14	29	37	13

Copy 37

Insertion Sort

29	10	14	37	13

10	14	29	37	13

There are no bigger ones on the left side

Insertion Sort

29	10	14	37	13
10	14	29	37	13

Leave 37 to its original place

Insertion Sort

29	10	14	37	13

10	14	29	37	13

Last value, 13

Insertion Sort

29	10	14	37	13

10	14	29	37	13

Copy 13

Insertion Sort

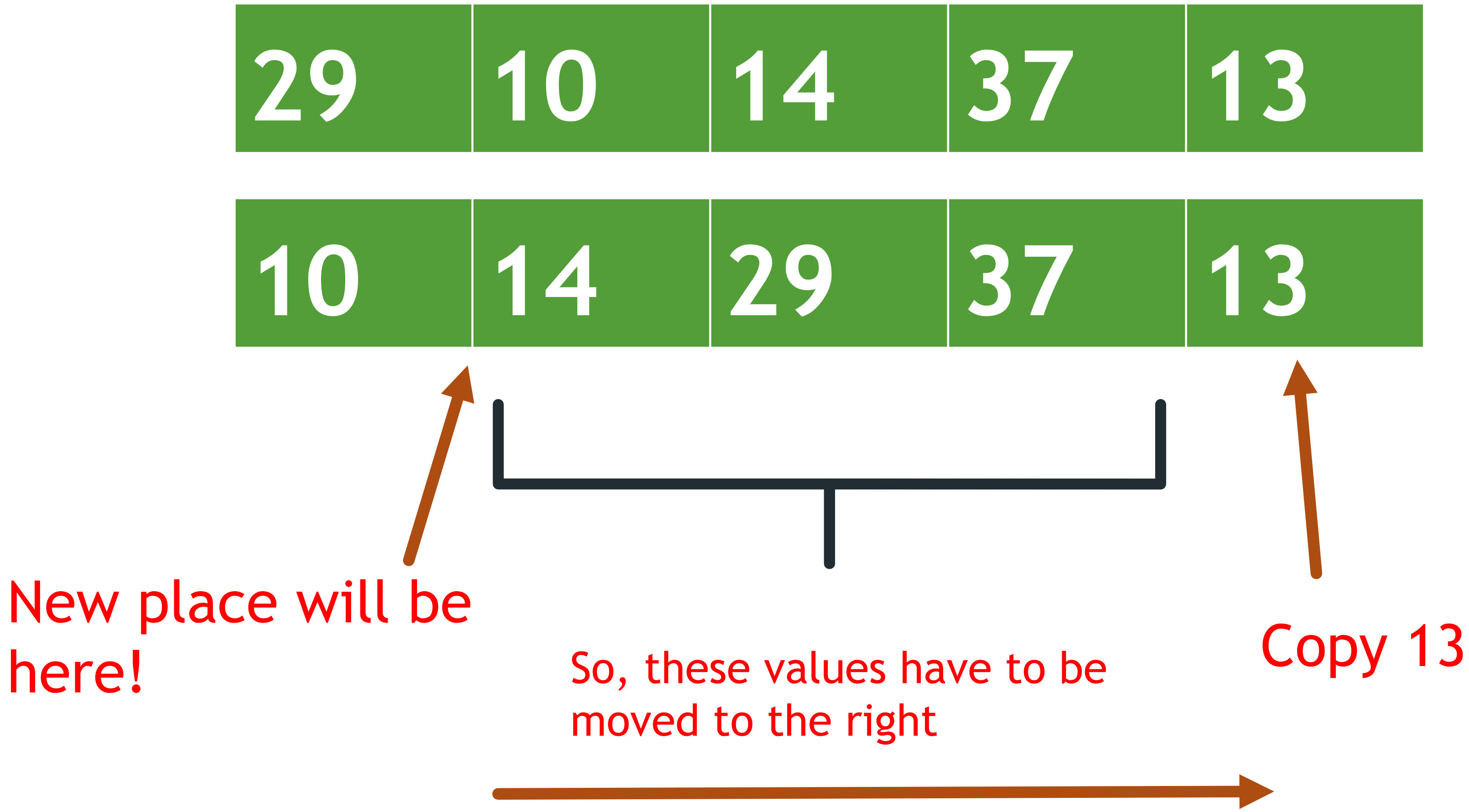
29	10	14	37	13

10	14	29	37	13

New place will be here!

Copy 13

Insertion Sort



Insertion Sort

29	10	14	37	13

10	14	14	29	37

Move value 14, 29 and 37 to the right

Insertion Sort

29	10	14	37	13

10	13	14	29	37

Add 13

Insertion Sort

29	10	14	37	13

10	13	14	29	37

Ready!

Insertion Sort

29	10	14	37	13

10	13	14	29	37

How efficient is this algorithm?
Time complexity,
$T(n) = O(n^2)$,
where n is array size.
When n grows, elapsed running time follows function $f(n^2)$.

Java code

```
static void inssort(int array[])
{
 int amount = array.length;
 int i, temp, pos, min, newValue, newPlace, currentPlace;
 min = array[0];
 pos = 0;

 for (i = 0; i < amount; i++)
   if (array[i] <= min)
   {
     min = array[i];
     pos = i;
   }

  temp = array[0];
  array[0] = min;
  array[pos] = temp;

  for (newPlace = 1; newPlace < amount; newPlace++)
  {
    newValue = array[newPlace];
    currentPlace = newPlace;
    while (array[currentPlace - 1] > newValue)
    {
      array[currentPlace] = array[currentPlace - 1];
      currentPlace = currentPlace - 1;
    }
    array[currentPlace] = newValue;
  }
}
```

Java code: test

```
public static void main(String[] args) {

    int[] vals = {10,14,29,37,13};

    inssort(vals);
    int amount = vals.length;

     for (int i = 0; i < amount; i++)
         System.out.println(vals[i]);

}
```

```
run:
10
13
14
29
37
```

Quick Sort

4	2	3	1	4	1	8	7	6	5

Easy learning, pale info in a nutshell!

Quick Sort

4	2	3	1	4	1	8	7	6	5

1) Find the pivot value

Quick Sort

4	2	3	1	4	1	8	7	6	5

1) Find the pivot value
   * it can be first value
   * it can be the median
   * it is good choose the median of 3 values: first, last, middle

Quick Sort

4	2	3	1	4	1	8	7	6	5

1) Find the pivot value
   * it can be first value
   * it can be the median
   * it is good choose the median of 3
     values: first, last, middle

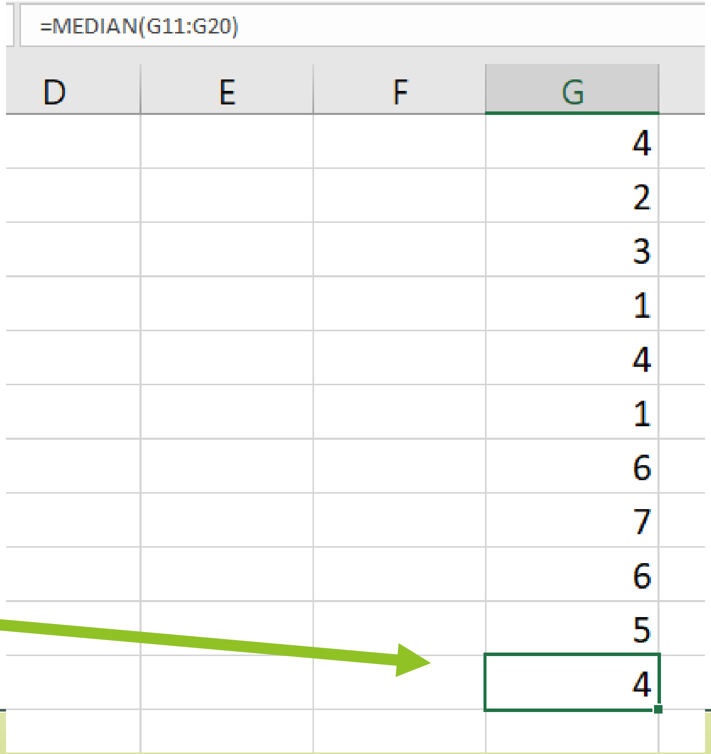
Now, let's use median!

Quick Sort

4	2	3	1	4	1	8	7	6	5

1) Find the pivot value
   * it can be first value
   * it can be the median
   * it is good choose the median of 3
     values: first, last, middle

Now, let's use median!


=MEDIAN(G11:G20)

D	E	F	G
			4
			2
			3
			1
			4
			1
			6
			7
			6
			5
			4

Excel gives median 4.

Quick Sort

4	2	3	1	4	1	8	7	6	5

1) Find the pivot value
   * it can be first value
   * it can be the median
   * it is good choose the median of 3
     values: first, last, middle

SO, first pivot value is 4

Quick Sort

4	2	3	1	4	1	8	7	6	5

SO, now we divide our array to 2 parts: values that are smaller than pivot and values that are bigger than pivot

Quick Sort

4	2	3	1	4	1	8	7	6	5

SO, now we divide our array to 2 parts: values that are smaller than pivot and values that are bigger than pivot

4	2	3	1	4	1

8	7	6	5

Quick Sort

4	2	3	1	4	1	8	7	6	5

SO, now we divide our array to 2 parts: values that are smaller than pivot and values that are bigger than pivot

4	2	3	1	4	1

8	7	6	5

New pivot values: 3 and 6

Quick Sort

4	2	3	1	4	1	8	7	6	5

SO, now we divide our array to 2 parts: values that are smaller than pivot and values that are bigger than pivot

Pivot is 3

4	2	3	1	4	1

2	1	1

4	3	4

Quick Sort

4	2	3	1	4	1	8	7	6	5

SO, now we divide our array to 2 parts: values that are smaller than pivot and values that are bigger than pivot

Pivot is 6

8	7	6	5

5

8	7	6

Pivot is 7

6

8	7

Quick Sort

4	2	3	1	4	1	8	7	6	5

SO, now we divide our array to 2 parts: values that are smaller than pivot and values that are bigger than pivot

Pivot is 3

4	2	3	1	4	1

2	1	1

4	3	4

Pivot is 4

Pivot is 2

1	1

2

3

4	4

Quick Sort

4	2	3	1	4	1	8	7	6	5

Now, we sort all partial arrays and combine them to form a sorted array!

Quick Sort

4	2	3	1	4	1	8	7	6	5

Code

```
void sort(int first, int last, int array[])
{
int start, left, right, temp;
left = first;
right = last;
start = array[(first+last)/2];

do
{
while (array[left] < start)
left= left +1;
while (start < array[right])
right = right - 1;
if (left <= right)
{
swap (&(array[left]), &(array[right]));
left= left + 1;
right = right - 1;
}
}
while ((right > left));
if (first < right) sort(first,right, array);
if (left < last) sort(left, last, array);
}
```

Quick Sort

4	2	3	1	4	1	8	7	6	5

Test run

```
int main()
{
    int values[] = {4,2,3,1,4,9,8,7,6,5};
    sort(0, 9, values);

    for (int k = 0; k < 10; k++)
        cout << values[k] << endl;
}
```

```
1
2
3
4
4
5
6
7
8
9
```

Quick Sort

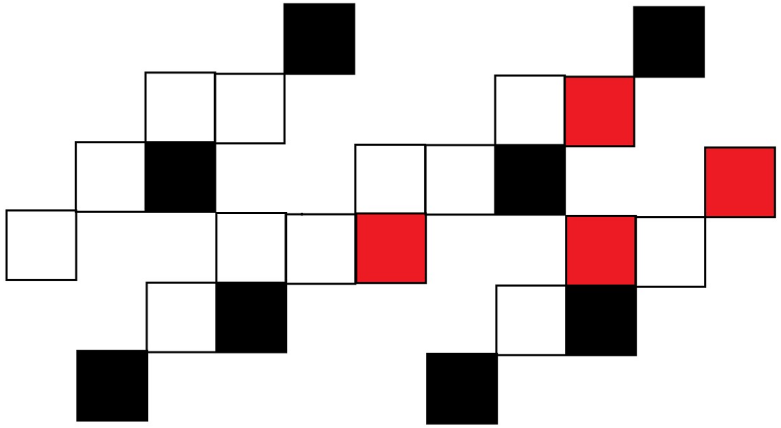
Sorting time example

10 millions values -> 2 seconds!

```
int main()
{
        int size = 10000000;
        int * values = new int[size];
        for (int k = 0; k < size; k++)
        {
            values[k] = rand();
        }

        long t1 = time(NULL);
        sort(0, size-1, values);
        long t2 = time(NULL);

       cout << "It took " << (t2 - t1)  << " secs \n";
```

```
It took 2 secs

---------------
Process exited
Press any key t
```

Shell Sort



Simulating sorting methods

Shell Sort

20	30	5	9	2	0	22

Shell sort is a slow sorting method but it is normally faster than selection sort:
* many comparisons and swappings but no so many as in selection sort
* now we compare elements using distances

Now we are going to simulate shell sort!

Shell Sort

20	30	5	9	2	0	22

Definition of this array can be like this:

```
int values[] = {20, 30, 5, 9, 2, 0, 22};
```

Shell Sort

Definition of this array:
int values[] = {20, 30, 5, 9, 2, 0, 22};

20	30	5	9	2	0	22

Round 1:
First distance between elements that are compared to each other is normally size of the array divided by:
now it is 7/2 and we can round it to be 3.
We want to find the smallest value and add it to the beginning of this array.
SO, the first value is now 20, place is values[0].
Now we compare 20 to the value that is 3 places from place 0, and it is place 3 and there we have value 2.

Shell Sort

Definition of this array:
int values[] = {20, 30, 5, 9, 2, 0, 22};

20	30	5	9	2	0	22

Round 1:
SO, let's go on:
2 < 20?
Yes, we swap values and get:

2	30	5	9	20	0	22

Shell Sort

Definition of this array:
int values[] = {20, 30, 5, 9, 2, 0, 22};

2	30	5	9	20	0	22

Round 1 goes on:
We go on with value 30 now:
0 < 30?
Yes, values are swapped and we get

2	0	5	9	20	30	22

Shell Sort

Definition of this array:
int values[] = {20, 30, 5, 9, 2, 0, 22};

2	0	5	9	20	30	22

Round 1 goes on:
We go on with value 2 now:
22 < 5?
No, we do nothing
So, after 1. round we have situation:

2	0	5	9	20	30	22

Shell Sort

Definition of this array:
int values[] = {20, 30, 5, 9, 2, 0, 22};

2	0	5	9	20	30	22

Round 2:
Distance is now 3/2, we round it to 2
9 < 2?
No
20 < 0?
No
30 < 5?
No
22 < 9?
No

Shell Sort

Definition of this array:
int values[] = {20, 30, 5, 9, 2, 0, 22};

2	0	5	9	20	30	22

Round 3: distance is now 1
5 < 2?
No
9 < 0?
No
20 < 5?
No
30 < 9?
No
22 < 20?

Shell Sort

Definition of this array:
int values[] = {20, 30, 5, 9, 2, 0, 22};

2	0	5	9	20	30	22

Round 4: distance is now 0
0 < 2?
Yes

0	2	5	9	20	30	22

5 < 2?
9 < 5?
20 < 9?
30 < 20?
22 < 30?
Yes, swapping

Shell Sort

Definition of this array:
int values[] = {20, 30, 5, 9, 2, 0, 22};

0	2	5	9	20	22	30

That’s is!

Array is sorted!

Shell Sort

Here is c code:

```
void shell(int values[], int size)
{
  int k, distance, swap = 1;
  distance = size / 2;
   do
    do
    {
        swap = 0;
        for (k = 0; k < (size - distance); k++)
        if (values[k] > values[k + distance])
        {
          int temp = values[k];
          values[k] = values[k + distance];
          values[k + distance] = temp;
          swap = 1;
        }
    } while (swap == 1);
 while ( (distance /= 2) > 0);
}
```

Shell Sort

Let's try using different input sizes
Here is c code
a) filling array

```
int size = 20000000;
int * values = calloc(size, 4);
int i;
for (i = 0; i < size; i++)
{
    values[i] = rand() % 10000;  // values 0 - 9999 assigned
}
```

Shell Sort

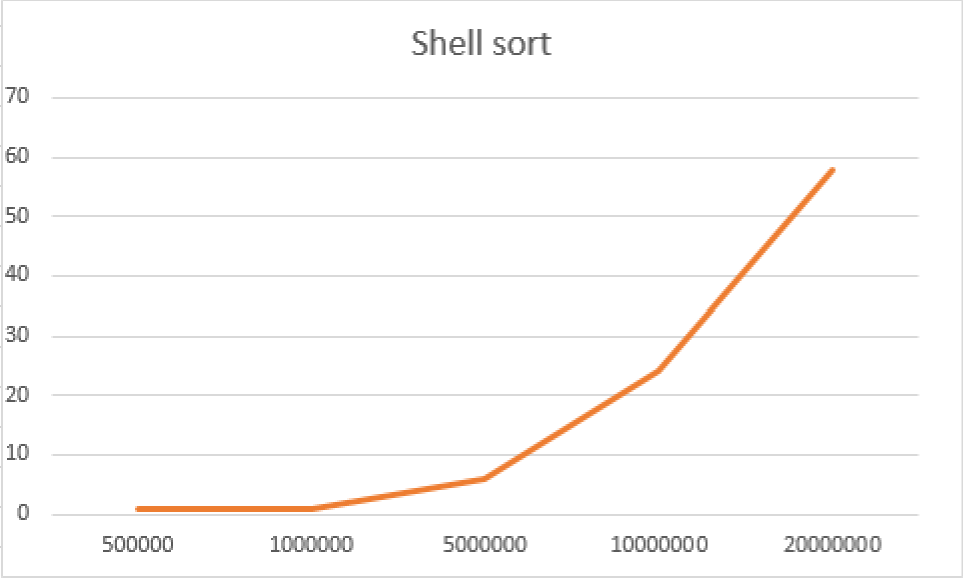
Let's try using different input sizes
Here is c code
b) taking execution time

```
int time1 = time(0);

shell(values, size);

int time2 = time(0);
int time_elapsed = time2 - time1;

printf("\n\n\nIt took %d secs \n\n\n", time_elapsed);
```

Selection Sort

Execution times as a diagram

500000	1
1000000	1
5000000	6
10000000	24
20000000	58



Thank You!

Give feedback!

This ebook copy is free!