OSS CS50 - week 3

Linary search

```
for each element in array
     If element you're looking for
          Return true
return false
```

Binary search

- Divide and conquer
  Divide in half, pick half that it has to be in, divide in half, etc.

```
look in middle of array
if element you're looking for
     return true
else if element is to left
     search left half of array
else if element is to right
     search right half off array
else
     return false
```

Bubble sort

- Swap variables pair-wise

```
repeat until no swaps
     for i from 0 to n-2
          if i'th and i+1'th elements out of order
               swap them
```

Selection sort

```
for i from 0 to n-1
     find smallest element between i'th and n-1'ith
     swap smallest with i'th element
```

Insertion sort

- Deal with variable one-by-one

```
for i from 1 to n-1
      call 0'th through i-1'th elements the "sorted side"
      remove i'th element
      insert it into sorted side in order
```

Running time

Bubble sort (n-1) + (n-2) + …. + 1 =
n(n-1) / 2
$(n^2 - n)/2$
$n^2/2 - n/2$

n = 1,000,000
$n^2/2 - n/2$ = 499,999,500,000 ≅ 5 billion

$\boldsymbol{O}(n^2)$ = 5 billion

notation ***O*** (big O) = running time (on the order of)

Running time of bubble sort, insertion sort and selection sort all are $\boldsymbol{O}(n^2)$

Big O - ***O*** (running time)

$\boldsymbol{O}(n^2)$	- bubble sort, selection sort, insert sort
$\boldsymbol{O}(n \log n)$	
$\boldsymbol{O}(n)$	- linear search
$\boldsymbol{O}(\log n)$	- binary search
$\boldsymbol{O}(1)$	*constant time* (1 step, e.g. printf, check if)

www.bigocheatsheet.com

Big Omega ***Ω*** (lower bound)

$\boldsymbol{\Omega}(n^2)$	- selection sort
$\boldsymbol{\Omega}(n \log n)$	
$\boldsymbol{\Omega}(n)$	- bubble sort
$\boldsymbol{\Omega}(\log n)$	
$\boldsymbol{\Omega}(1)$	

if Big-O and Big-Ω are the same, the running time is $\boldsymbol{\Theta}$ (Capital Theta).

Visualization

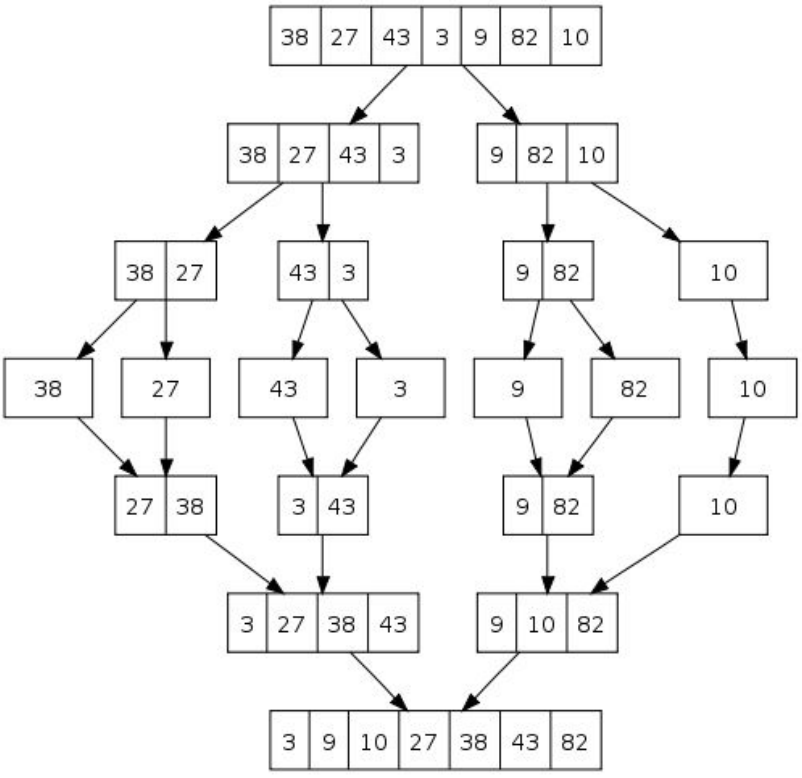
https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

Merge sort

```
on input of n elements
      if n < 2
            return 2
      else
            sort left half of elements[a]
            sort right half of elements[b]
            merge sorted halves[b]
```

$T(n) = T(n/2)^a + T(n/2)^b + O(n)^c$

(if n ≥ 2)



Running time = ***O***(n log n)

Recursion

```
#include <cs50.h>
#include <stdio.h>

int sigma(int m);

int main(void)
{
    int n;

    do
    {
        printf("Positive integer please: ");
        n = get_int();
    }
    while (n < 1);

    int answer = sigma(n);

    printf("%i\n", answer);
}

int sigma(int m)
{
    if (m <= 0)
    {
        return 0;
    }
    else
    {
        return (m + sigma(m - 1));
    }
}
```