Analyzing Algorithm Examples

General Rules of analyzing algorithm code:

Rule 1 — *for* loops:

The running time of a **for** loop is at most the running time of the statements inside the **for** loop (including tests) **times** the number of iterations.

Rule 2 — Nested loops:

Analyze these **inside out**. The total running time of a statement inside a group of nested loops is the running time of the statement multiplied by the product of the sizes of all the loops.

Rule 3 — Consecutive Statements:

These just add (which means that the maximum is the one that counts.

Rule 4 — *if/else*:

```
if( condition )
    S1
else
    S2
```

The running time of an **if/else** statement is never more than the running time of the **test** plus the larger of the running times of **S1** and **S2**.

Rule 5 — *methods call*:

If there are method calls, these must be analyzed first.

Sorting Algorithm

1- Bubble Sort (revision) ➔ O(n^2)

```java
public static void bubble(int[] arr){
  int temp;
  for (int i = 0; i < arr.length-1; i++) {
    for (int j = 0; j < arr.length-i-1 ; j++) {
      if(arr[j+1]<arr[j]){
        temp = arr[j];
        arr[j] = arr[j+1];
        arr[j+1] = temp;
      }
    }
  }
}
```

2- **Selection Sort → O(n^2)**: named selection because every time we select the smallest item.

```
public static void selection (int[] arr){
   int temp, minIndex;
   for (int i = 0; i < arr.length-1; i++) {
      minIndex = i;
      for (int j = i+1; j <arr.length ; j++) {
         if(arr[j]<arr[minIndex]){
            minIndex=j;
         }
      }
      if(i!= minIndex){
         temp = arr[i];
         arr[i] = arr[minIndex];
         arr[minIndex] = temp;
      }
   }
}
```

3- **Insertion sort → O(n^2):**

```
public static void insertion (int[] arr){
   int j, current;
   for (int i = 1; i < arr.length; i++) {
      current = arr[i];
      j=i-1;
      while (j>=0 && arr[j]>current){
         arr[j+1] = arr[j];
         j--;
      }
      arr[j+1]=current;
   }
}
```

O(n^2) sorting algorithms comparison:

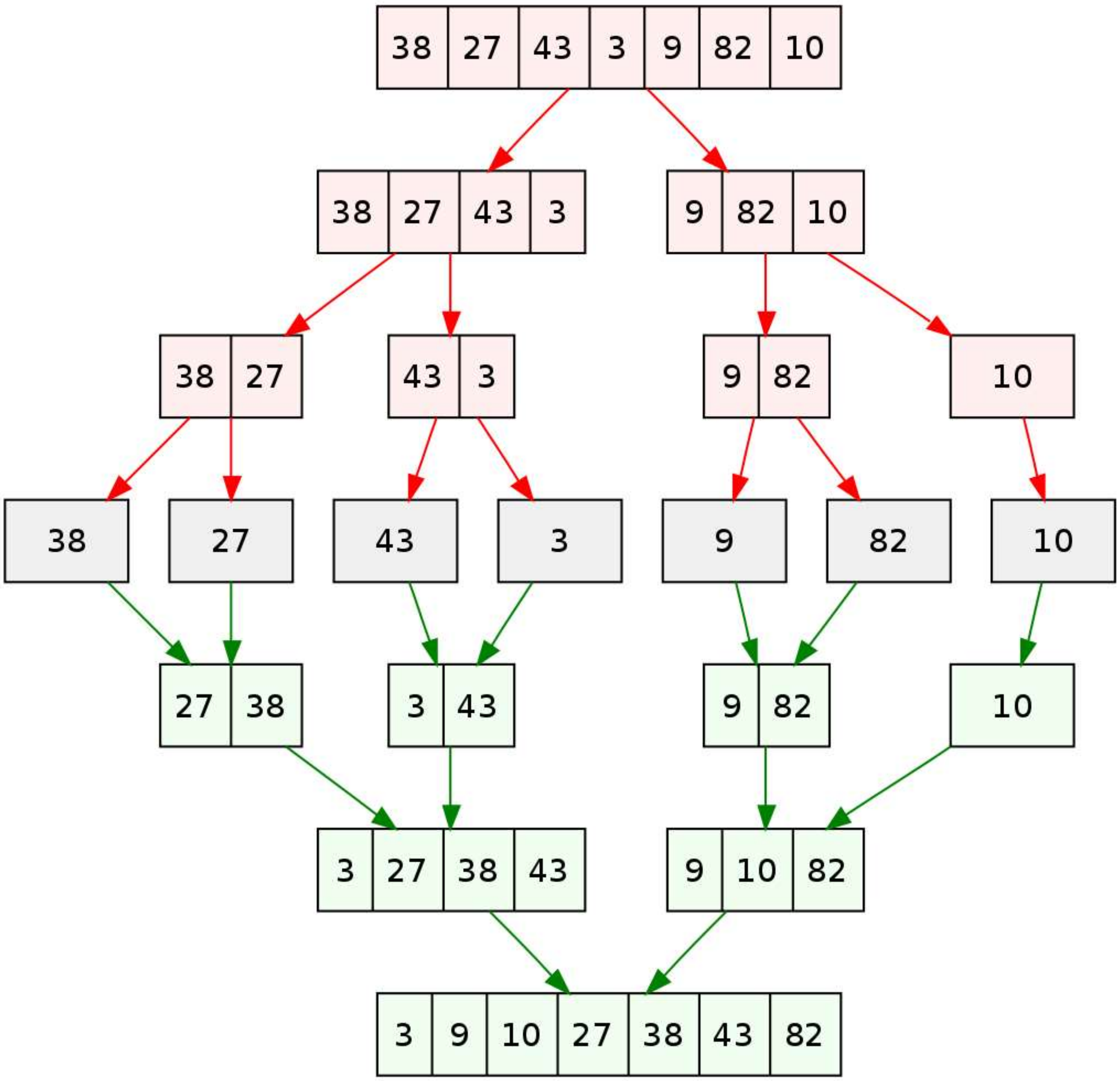
(run demo @ http://www.sorting-algorithms.com/)

Bubble Sort	Selection Sort	Insertion Sort
Very inefficient	• Better than bubble sort • Running time is independent of ordering of elements	• Relatively good for small lists • Relatively good for partially sorted lists

Merge sort: recursive algorithm

Merge: take 2 sorted arrays and merge them together into one.

Step 1 − if it is only one element in the list it is already sorted, return.
Step 2 − divide the list recursively into two halves until it can no more be divided.
Step 3 − merge the smaller lists into new list in sorted order.
Example:



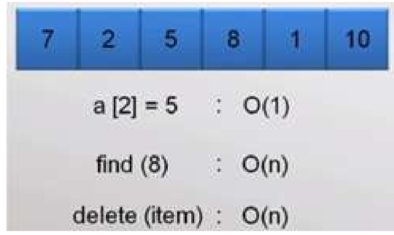
Java code:

```java
void sort(int nums[], int left, int right) {
        if (left < right) {
                int m = (left + right) / 2;// Find the middle point
                sort(nums, left, m); // Sort first halve
                sort(nums, m + 1, right); // Sort second halve
                merge(nums, left, m, right); // Merge the sorted halves
        }
}

void merge(int nums[], int left, int m, int right) {
        int n1 = m - left + 1;
        int n2 = right - m;

        int Left_arr[] = new int[n1];
        int Right_arr[] = new int[n2];

        for (int i = 0; i < n1; ++i)
                Left_arr[i] = nums[left + i];
        for (int j = 0; j < n2; ++j)
                Right_arr[j] = nums[m + 1 + j];

        int i = 0, j = 0;

        int k = left;
        while (i < n1 && j < n2) {
                if (Left_arr[i] <= Right_arr[j]) {
                        nums[k] = Left_arr[i];
                        i++;
                } else {
                        nums[k] = Right_arr[j];
                        j++;
                }
                k++;
        }

        while (i < n1) {
                nums[k] = Left_arr[i];
                i++;
                k++;
        }

        while (j < n2) {
                nums[k] = Right_arr[j];
                j++;
                k++;
        }
}
```

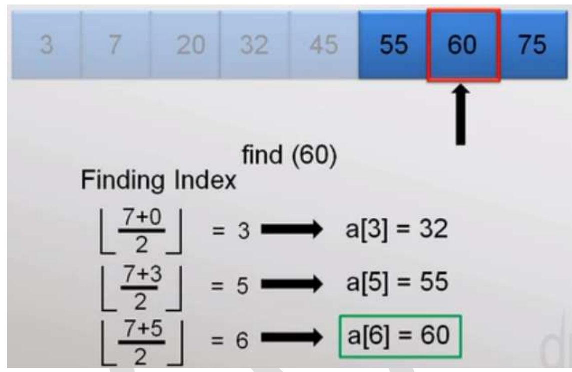
H.W: implement merge sort your own

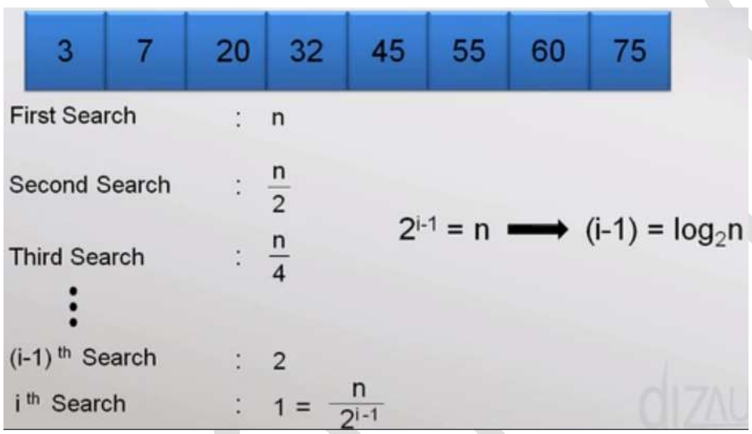
Searching elements in an array:

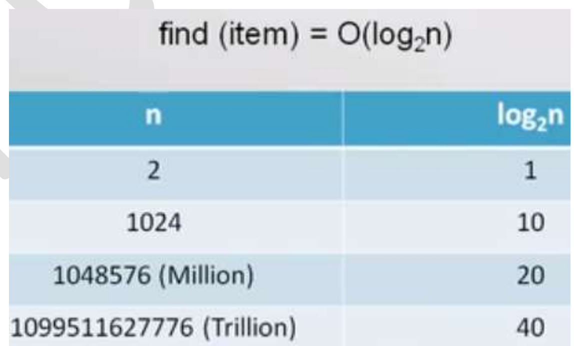
Case 1: unordered array:



Case 2: ordered array: -Binary search-







find (item) = $O(\log_2 n)$

n	$\log_2 n$
2	1
1024	10
1048576 (Million)	20
1099511627776 (Trillion)	40

Inserting and deleting items from ordered array