Sorting and Searching Algorithms

Selection Sort:

- Selection Sort is a simple sorting algorithm that repeatedly finds the minimum element from the unsorted part of the array and puts it at the beginning.
- It maintains two subarrays: sorted and unsorted. Initially, the sorted subarray is empty, and the unsorted subarray is the entire array.
- The algorithm iterates through the unsorted subarray, finds the minimum element, and swaps it with the first element of the unsorted subarray. This effectively grows the sorted subarray by one element.
- The process is repeated until the entire array is sorted.

Steps:

a. Start with an unsorted array of elements.

b. Repeat the following steps until the entire array is sorted:

i. Find the minimum element in the unsorted part of the array.

ii. Swap the minimum element with the first element of the unsorted part of the array.

iii. Move the boundary of the sorted subarray one element to the right.

c. When the entire array is sorted, the algorithm terminates.

Example 1

https://www.programiz.com/dsa/selection-sort

Example 2

{64, 25, 12, 22, 11}

Example 3

https://www.simplilearn.com/tutorials/data-structure-tutorial/selection-sort-algorithm

```
// Function for Selection sort
void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

    for (i = 0; i < n - 1; i++) {

        min = i;
        for (j = i + 1; j < n; j++) {
            if (arr[j] < arr[min])
                min = j;
        }

        if (min != i)
            swap(arr[min], arr[i]);
    }
}
```

Insertion Sort

Step 1 - If the element is the first element, assume that it is already sorted.

Step 2 - Pick the next element, and store it separately in a **key.**

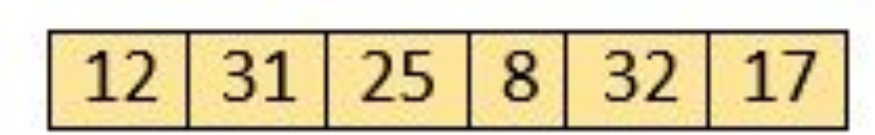
Step 3 - Now, compare the **key** with all elements in the sorted array.

Step 4 - If the element in the sorted array is smaller than the current element, then move to the next element. Else, shift greater elements in the array towards the right.

Step 5 - Insert the value at the proper position

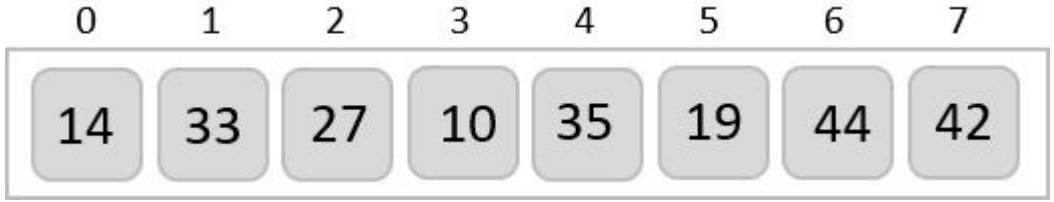
Step 6 - Repeat until the array is sorted.

Example 1



https://www.javatpoint.com/insertion-sort

Example 2



https://www.tutorialspoint.com/data_structures_algorithms/insertion_sort_algorithm.htm

```
void insertionSort(int arr[], int n) {

    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}
```

Binary Search

Binary Search is a searching algorithm for finding an element's position in a sorted array.

In this approach, the element is always searched in the middle of a portion of an array.

Binary search can be implemented only on a sorted list of items. If the elements are not sorted already, we need to sort them first.

Binary Search Working

1.The array in which searching is to be performed is:

Let x = 4 be the element to be searched.

2. Set two pointers low and high at the lowest and the highest positions respectively.

3. Find the middle element mid of the array ie. arr[(low + high)/2] = 6

4. If x == mid, then return mid.Else, compare the element to be searched with m.

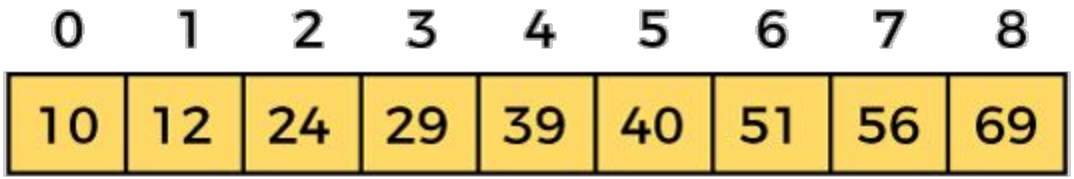
5. If x > mid, compare x with the middle element of the elements on the right side of mid. This is done by setting low to low = mid + 1.

6. Else, compare x with the middle element of the elements on the left side of mid. This is done by setting high to high = mid - 1.

7. Repeat steps 3 to 6 until low meets high.

8. x = 4 is found

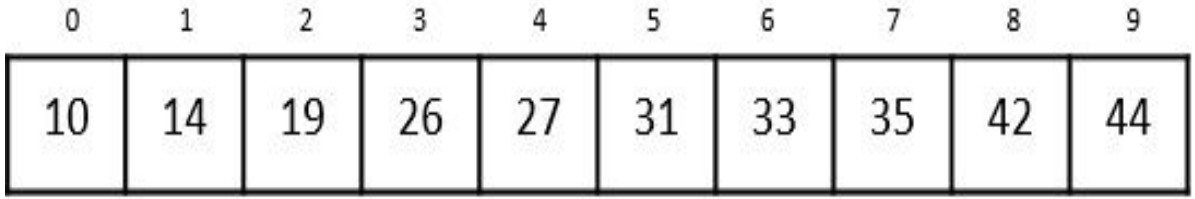
Example 2



Search 40

https://www.javatpoint.com/binary-search

Example 3



Search 100

https://www.tutorialspoint.com/data_structures_algorithms/binary_search_algorithm.htm

```
int binarySearch(int arr[], int l, int r, int x)
{
    while (l <= h) {
        int m = l + h / 2;

        if (arr[m] == x)
            return m;

        if (arr[m] < x)
            l = m + 1;

        else
            h = m - 1;
    }

    return -1;
}
```