Searching & Sorting

Stefan Zorcic

Searching:

Searching is an important part of programming, given a list of values find the index of a certain value, the application for searching are endless for example to get to this website you had to Google the link (which most likely used a searching algorithm). This tutorial will cover the two most prevalent searching algorithms: sequential search and binary search.

Sequential search is the easiest search to implement however it is has a time complexity of O(n) which is the best time complexity for an unsorted array however often too slow to be used in many professional applications. Sequential search iterates over the array and compares to the value to be found and returns index in element is in array and -1 otherwise.

```
public static int search(int [] arr, int value){
   for (int i=0;i<arr.length;i++){
      if (arr[i]==value){
         return i;
      }
   }
   return -1;
}
```

Binary search is significantly faster than sequential search with a time complexity of O (log n) however binary search requires the array to be sorted to properly work. Therefore Binary search is typically used over sequential search if the array needs to be searched for multiple values and vice versa otherwise. Binary search checks the middle value of the array and if the value is greater than the value to be found it discard the right, otherwise discarding the left. Binary search repeats this process until the element is found.

```
int binarySearch(int arr[], int val)
 {
     int left = 0;
     int right= arr.length - 1;

     while (left <= right) {
         int mid= left + (right - left) / 2;

         if (arr[m] == val){ return m; }

         if (arr[m] < val){ left = mid+ 1; }

         else { right = mid- 1; }
     }

     return -1;
 }
```

Sorting:

Sorting like searching has a plethora of application, in fact, this tutorial has already covered a uses with binary search. Sorting is considered to have a cost of O (n log n). However this time complexity is reserved for the advanced sorts. This tutorial will cover: bubble sort, insertion sort, selection sort, merge sort.

Bubble sort has a time complexity of O (n^2) and hence is not used on account of the algorithm being slow. Bubble sorts operates comparing a pair of elements and swapping if needed, repeating this process for the entire array. A implementation of bubble sort is below.

```
public void bubbleSort(int arr[])
{
     int len = arr.length;
     for (int i = 0; i < len-1; i++)
         for (int z = 0; z < len-i-1; z++)
             if (arr[z] > arr[z+1])
             {
                 int temp = arr[z];
                 arr[z] = arr[z+1];
                 arr[z+1] = temp;
             }
}
```

Insertion sort like bubble sort is not utilized because of its time complexity of O (n^2). Insertion sort's algorithm is similar to bubble sort as the algorithm iterates over the array and compares the current element to the previous element and swap accordingly.

```
public void insertionSort(int arr[])
{
     int len = arr.length;
     for (int i = 1; i < len; ++i) {
         int val = arr[i];
         int z = i - 1;
         while ((z >= 0) && (arr[z] > val)) {
             arr[z + 1] = arr[z];
             z -= 1;
         }
         arr[z + 1] = val;
     }
}
```

Selection sort like the previous sorts has a time complexity of O (n^2) and is not frequently used because it is relatively slow. Selection sort finds the minimum value element in the unsorted array and adds it to the front of the unsorted section and labels that point sorted. Selection sort repeats this process for the entire array.

```
public void selectionSort(int arr[])
 {
     int len = arr.length;
     for (int i = 0; i < len-1; i++)
     {
         int index = i;
         for (int z = i+1; z < len; z++){
             if (arr[z] < arr[index]){
                 index = z;
             }
         }

         int temp = arr[index];
         arr[index] = arr[i];
         arr[i] = temp;
     }
 }
```

Merge sort with a time complexity of O (n log n) is the sorting algorithm that is used primarily is most professional application. However merge sort is the most complex of all the sorting algorithm presented so far. Merge sort follows these basic recursive steps:

mergeSort function:

1. Find the middle point to divide the array into two halves:
2. Call mergeSort for first half:
3. Call mergeSort for second half:
4. Merge the two halves sorted in step 2 and 3: