Searching and Sorting (Practice Set-1)

(Courtesy Aditya Pancholi)

1. Consider the following code and answer the questions below

```
my_Search(arr, num_ele, key)
        if num_ele < 1
            return -1

        else
            if key == arr[num_ele-1]
                return (num_ele-1)

            else
                return my_search(arr,num_ele-1,key)
```

   (a) What does the above function perform?

   (b) What is the best case and worst case input for the above function?

   (c) Write the recurrence relation for the above function.

   (d) What is the running time for the above function?

2. Consider the following code and answer the questions below

```
my_Search(arr, low, high, key)
        if low > high
            return False

        else
            mid = (low + high) / 2
                if key == arr[mid]
                return True

            else
                found1 = my_Search(arr, low, mid-1, key)
                found2 = my_Search(arr, mid + 1, high, key)
                return found1 OR found2
```

   (a) What does the above function perform?

   (b) What is the best case and worst case input for the above function?

   (c) Write the recurrence relation for the above function.

   (d) What is the running time for the above function?

3. Consider the following code and answer the questions below

```
my_Search(arr, low, high, key)
        if low > high
            return False

        else
            mid = (low + high) / 2
                if key == arr[mid]
                return True

            else if key > arr[mid]
                return my_Search(arr, mid + 1, high, key)

            else
                return my_Search(arr, low, mid - 1, key)
```

(a) What does the above function perform?

(b) What is the best case and worst case input for the above function?

(c) Write the recurrence relation for the above function.

(d) What is the running time for the above function?

(e) If the statement mid=(low+high)/2 is replaced by **mid=(low+high)/3**

1. What is the best case and worst case input for the above function?
2. Write the recurrence relation for the best case time complexity of the above function.
3. Write the recurrence relation for the worst case time complexity of the above function.
4. What is the running time for the above function?

(f) If the statement mid=(low+high)/2 is replaced by **mid=2*(low+high)/3**

1. What is the best case and worst case input for the above function?
2. Write the recurrence relation for the best case time complexity of the above function.
3. Write the recurrence relation for the worst case time complexity of the above function.
4. What is the running time for the above function?

(g) If the statements *return my_Search(arr, mid + 1, high, key)* and *return my_Search(arr, low, mid-1, key)* are replaced by **return my_Search(arr, mid , high, key)**

and **return my_Search(arr, low, mid, key)** respectively, What effect will it have on the function?

4. Given a sorted array $A[1, \ldots, n]$ such that elements may not be distinct. Modify binary search such that for a given key, it returns the index of the first occurrence of key. Also analyze the running time of your modified algorithm.

5. Given a sorted array $A[1, \ldots, n]$ such that elements are distinct. The array is rotated around some random position.
   (a) Modify binary search such that for a given key, it returns the index where key is present (if it exists).
   For example if the input array is $[5, 8, 10, 11, -2, 0, 1, 2, 4]$ and $key = 10$, it returns the index where 10 is present, that is 3.
   (b) Can your algorithm be further modified to handle the case where elements may not be distinct?

6. Given a sorted array $A[1, \ldots, n]$ such that elements may not be distinct. Give an $\mathcal{O}(lg(n))$ algorithm that returns the frequency of a given element key.

7. Given a sorted array $A[1, \ldots, n]$ such that elements are all distinct. Give an $\mathcal{O}(lg(n))$ algorithm that finds out if $\exists k$ such that $A[k] = k$. The algorithm returns k if such an index exists, else -1.

8. A ternary search is similar to binary search where instead of splitting the array into 2 equal parts, we split the array into 3 equal parts.
   (a) Write the algorithm to search a given key using Ternary Search.
   (b) Write the recurrence relation for the best case time complexity of the above algorithm.
   (c) Write the recurrence relation for the worst case time complexity of the above algorithm.
   (d) Solve the recurrence relation to calculate the running time of Ternary Search.
   (e) Compare the running time of Ternary Search to standard Binary Search.

9. You are given two sorted lists of size m and n respectively. Give an $\mathcal{O}(lg(m)+lg(n))$ algorithm for computing the k^{th} smallest element in the union of two lists.

10. Suppose if we modify Insertion Sort as follows:
    For each element $Arr[i]$ for i in range $2..n$, we search the correct location in the sorted sub-array $Arr[1..(i-1)$ using **Binary search** (instead of Linear Search) and then insert it at its correct location.

(a) Write the best case and worst case recurrence relation for your modified Insertion Sort.

(b) Calculate the best case and the worst case running time of your modified Insertion Sort. Does this modification improve the performance of Insertion Sort?

(c) Give the best case and worst case inputs for your modified Insertion Sort.

11. Suppose you are given the *head* pointer of a singly linked list where each node contains an integer. Give an algorithm to sort this linked list using Insertion-Sort.

(a) What is the best case and worst case input for your algorithm?

(b) Write the best case and the worst case recurrence relation for your algorithm?

(c) What is the the running time for your algorithm?

(d) Can the running time of Insertion Sort be improved if we had a doubly linked list instead of singly linked list?

12. Suppose you are given the *head* pointer of a singly linked list where each node contains an integer. Give an algorithm to sort this linked list using Merge-Sort.

(a) Write the recurrence relation for your algorithm?

(b) What is the the running time for your algorithm?

13. Consider the modification to Merge Sort where as soon as the subarray reduces to size k or smaller, instead of sorting it using Merge Sort, we sort it using Insertion Sort instead. Give the running time of this modified Merge Sort.
Can you further reduce the running time of your algorithm by using Count Sort instead of Insertion Sort on sub problems of size k or smaller.

14. Given a sorted array $A[0 \ldots (n-1)]$ with distinct elements and an element *ele* to be searched. Consider the following searching algorithm.
Let $m = floor(\sqrt{n})$. We search the indices $0, m, 2m, 3m, \ldots$. Once we find the interval such that $A[k.m] < ele < A[(k+1).m]$, we perform linear search in that interval.

(a) Analyze the time complexity of this algorithm?

(b) Consider the case where array elements may not be distinct. Extend this algorithm to return the index for the first occurrence of the element *ele* in the array containing duplicates.

15. A Postal Index Number (PIN), refers to a six-digit code in the Indian postal code system used by India Post. Suppose you have n letters and you want to sort them

based on their PIN-code, suggest an efficient algorithm to do the same. (Assume n to be sufficiently large like $10,000$ letters)

16. Suppose you are given the *head* pointer of a singly linked list where each node contains an integer. Give an algorithm to sort this linked list using Quick-Sort.
    (a) Write the recurrence relation for your algorithm?
    (b) What is the the running time for your algorithm?