19CSE302: Design and Analysis of Algorithms

Lab Evaluation 1

Sorting and Graph Algorithms

CO01	Evaluate the correctness and analyze complexity of algorithms.
CO03	Design solutions for real world problems by identifying, applying and implementing appropriate design techniques.
CO05	Analyze the impact of various implementation choices on the algorithm complexity

All questions analysis to any AI tool like ChatGPT result to be added.

Question 1: Below is the algorithm for Quick Select (the Partition function is like that used in the Quick Sort algorithm). Implement the algorithm and run it for various test cases.
Answer the following questions:

1. Difference between Quick Sort and Quick Select algorithms.
2. Time complexity of the algorithms (Best/Worst Cases). Frame the recurrence relations for each of the cases and solve to compute the time complexity
3. List a few use cases of both algorithms
4. What are the different pivot selection methods. Give a demo of at least 3 methods with appropriate examples.
5. You have to split an unsorted integer array, W_n, of size n into the two subarrays, U_k and V_{n-k}, of sizes k and n − k, respectively, such that all the values in the subarray U_k are smaller than the values in the subarray V_{n-k}. You may use the following two algorithms:
   A. Sort the array W_n with QuickSort and then fetch the smallest k items from the positions i = 0, 1, . . . , k − 1 of the sorted array to form U_k (you should ignore the time of data fetching comparing to the sorting time).
   B. Directly select k smallest items with QuickSelect from the unsorted array (that is, use QuickSelect k times to find items that might be placed to the positions i = 0, 1, . . . , k − 1 of the sorted array).

Algorithm 1: QuickSelect

```
Data: a[ℓ : h]: the subarray a[ℓ], a[ℓ + 1], . . . , a[h] of an n-element
      array a. In the very first call, ℓ = 1 and h = n.
Result: The k-th largest element in a.
if ℓ = h then
    return a[ℓ]
end
else if ℓ < h then
    p ← choose the pivot, partition a[ℓ : h], and return the pivot's
      new index
    if k = p then
        return a[p]
    else if k < p then
        Apply QuickSelect to the left subarray a[ℓ : (p − 1)]
    else
        Apply QuickSelect to the right subarray a[(p + 1) : h]
    end
end
```

Choose the algorithm with the smallest processing time for any particular n and k. Quick Sort spends 10.24 microseconds to sort n = 1024 unsorted items and QuickSelect spends 1.024 microseconds to select a single i-smallest item from n = 1024 unsorted items. What algorithm, A or B, must be chosen if n = 2^20 and k = 2^9 ≡ 512?

Please complete your answer with the implementation of the algorithm and plotting the time complexity values for various input sizes.

Question 2: The Time complexity of sorting n data items with insertion Sort is $O(n^2)$; that is, the processing time of insertion Sort is $c_s \cdot n^2$ where c_s is a constant scale factor. The time complexity of merging k subarrays pre-sorted in ascending order and containing n items in total is O(k . n). The processing time is $c_m(k-1)\cdot n$ because at each step t = 1, 2, . . . , n you compare k current top items in all the subarrays (that is, (k − 1) comparisons), pick up the maximum item, and place it in position (n − t + 1) in the final sorted array. You save time by splitting the initial array of size n into k smaller subarrays, sorting each subarray by insertion Sort, and merging the sorted subarrays in a single sorted array. Let the scale factors c_s and c_m be equal: $c_s = c_m = c$. Analyse whether you can accelerate the sorting of n items in the following way:

- Split the initial array of size n into k subarrays of size n/k. The last subarray may be longer to preserve the total number of items n, but do not go into such detail in your analysis.
- Sort each subarray separately by Insertion Sort.
- Merge the sorted subarrays into a final sorted array. If the acceleration is possible, then find the optimum value of k giving the best acceleration and compare the resulting time complexity of the resulting sorting algorithm to that of Insertion Sort and Merge Sort.

Also submit the document including algorithm, time complexity and screenshot.

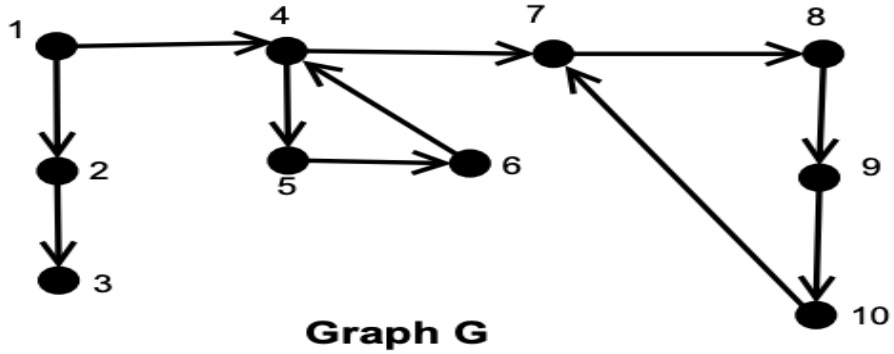
Question 3: Graphs

a. Let G be a connected weighted undirected graph with distinct edge weights. Prove that the following algorithm finds an MST of G. “Initially all the edges are unmarked. Let e be an unmarked edge with highest weight. If removing e does not disconnect the graph, remove it; otherwise, mark it. Repeat this process until the graph consists of marked edges only. Then, output the current graph as an MST of the original graph.”Also, submit the algorithm and time complexity.

b. Given below is the algorithm for identifying Strongly connected components. Find the SCC for the given graph. Your implementation should have function which computes transpose of the graph and print the Graph transpose

Algorithm 6 Strongly Connected Components (SCC)

Input: Directed Graph G
Output: Strongly Connected Components of G.
Step 1: Run DFS(G) and compute the finish time for all vertices.
Step 2: Find G^T and sort the vertex set of G in decreasing order based on its finish time.
Step 3: Run $DFS(G^T)$ from the vertex which has maximum finish time. Do this step repeatedly until all the vertices in G^T are visited at least once (this gives you the collection of SCC).
Step 4: Each forest in $DFS(G^T)$ is a SCC.



Graph G

Also submit the document including algorithm, time complexity and screenshot.

c. https://www.hackerearth.com/practice/algorithms/graphs/strongly-connected-components/practice-problems/algorithm/components-of-graph-2b95e067/

Solve the problem in Part c by modifying the solution to part b.