CS 6212 September 15, 2023
Youssef

Homework 2
Due Date: October 6, 2023

Problem 1: (10 points)

a. Show step by step how the heapsort algorithm sorts the array 27, 13, 56, 35, 48, 8, 18, 67, 5, 62, 7 starting from the min-heap you built in homework 1 (Problem 4a) for the same array.
b. Show step by step how the quicksort algorithm sorts the same array 27, 13, 56, 35, 48, 8, 18, 67, 5, 62, 7. Indicate at each step what the partitioning element is.
c. Show step by step how the mergesort algorithm sorts the same array.
d. Show step by step how the insertion sort algorithm sorts the same array. (This algorithm sorts by repeated insertion into a (initially empty) sorted array.)
e. Show step by step how selection sort sorts the same array. (This finds the minimum first, then the 2nd min, then the 3rd min, and so on. It has nothing to do with quick-select.)

Problem 2: (20 points)
Let $A[1:n]$ be an array of real numbers. The *distance* between two elements $A[i]$ and $A[j]$ is the absolute value $|A[i] - A[j]|$. Two elements $A[i]$ and $A[j]$ of A are called neighbors if $i = j - 1$ or $i = j + 1$. Two elements of A are called *the closest neighbors* if they are neighbors and the distance between them is the smallest distance between any two neighbors of A.

a. Write a divide-and-conquer algorithm that takes $A[1:n]$ and returns the following output: The minimum of A, the maximum of A, and the indices of the two closest neighbors in A. Note that there could be multiple closest neighbors; break the tie anyway you want.
b. Analyze the time complexity of your algorithm.

Note that for full credit, your algorithm should take less than $(n \log n)$.

Problem 3: (20 points)
Let $A[1:n]$ be an array of real numbers. A *subarray* of A is any sequence $A[i:j]$ made up of $A[i], A[i+1], \ldots, A[j]$ where $i \leq j$. The *sum* of the subarray $A[i:j]$ is the value $A[i] + A[i+1] + \cdots + A[j]$. We are interested to find the maximum-sum subarray, i.e., the subarray that has the largest sum in A.

a. Write a divide-and-conquer algorithm that takes $A[1:n]$ and returns maximum-sum subarray and its sum, that is, it returns the following output: (1) the starting index i, (2) the ending index j, and (3) the sum, of the maximum-sum subarray. In case of ties, break the tie arbitrarily.
b. Analyze the time complexity of your algorithm.

Note: For many divide-and-conquer algorithms, you may need to have the algorithm compute more outputs than initially stated, in order for the merge step to be doable efficiently.
Note: Again for full credit, your algorithm has to be as efficient as possible.

Problem 4: (10 points)
Consider this instance of the knapsack problem:

The weights are:	12,	8,	16,	4,	20,	28
The prices are:	48,	24,	80,	24,	40,	28
The capacity is:	M=28.					

Find the greedy the solution for this problem, showing the process step by step.

Problem 5: (20 points)
Let G=(V,E) be the following weighted graph:

V={1,2,3,4,5,6,7,8,9,10,11,12}

E={[(1,7) 2], [(1,8) 6], [(1,12) 5], [(7,12) 1], [(8,12) 2], [(12,6) 3], [(2,6) 7], [(2,7) 4], [(7,8) 7], [(12,5) 2], [(6,10) 5], [(5,10) 2], [(9,10) 1], [(9,4) 2], [(10,11) 2.5], [(11,4) 2.5], [(3,6) 2], [(3,9) 2.3]}, where [(*i*,*j*) *a*] means that (*i*,*j*) is an edge of weight *a*.

a. Use the greedy method (Kruskal's algorithm) to find a minimum spanning tree of G. Show the tree after every step of the algorithm.
b. Using the greedy Dijkstra's single-source shortest-path algorithm, find the distances between node 1 and all the other nodes. Show the values of the DIST array at every step.
c. Show the actual shortest paths of part (b). Note that these paths together form a spanning tree of G. Is this tree a minimum spanning tree?

Problem 6: (20 points)
You are given a list A of activities. Each activity a is represented by a pair (a_1, a_2) where a_1 is the start time of a, and a_2 is the end time of a. Your goal is to select the maximum number of non-overlapping activities that you can participate in, given that you can only participate in one activity at a time. We will use the greedy method for solving this problem.

a. Consider the following greedy selection policy: select the activity with the earliest starting time; in case of ties, select the one with the shortest duration among the tying activities. Prove by a counter-example that this method doesn't guarantee optimality.
b. Give a greedy policy that is guaranteed to yield an optimal solution, and prove optimality.
c. Illustrate your policy on $A = \{(1,5), (6,10), (3,8), (14,17), (9,13), (12,15), (7,11)\}$.

Bonus Problem: (5 points)
Let $A[1:n]$ be an array of real numbers. An *inversion* in A is a pair (i, j) where $i < j$ and $A[i] > A[j]$. A *split inversion* of A is an inversion (i, j) such that i is in the left half of A and j is in the right half of A.

a. Assume for now that A is made of two sorted halves. Write an algorithm that computes and returns the number of split inversions in A, and also merges the two halves into a sorted array. Analyze the time complexity of your algorithm.
b. Now, A is a totally arbitrary array of real numbers. Write a divide-and-conquer algorithm that sorts A and returns the total number of inversions. Analyze the time complexity of your algorithm.