CSE 431/531: Algorithm Analysis and Design Fall 2022

Homework 2

Instructor: Shi Li **Deadline: 10/9/2022**

Your Name: Zheyuan Ma Your Student ID: 50321597

Problems	1	2	3	Total
Max. Score	20	30	30	80
Your Score				

Problem 1 Construct the Huffman code (i.e, the optimum prefix code) for the alphabet $\{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}, \mathsf{f}, \mathsf{g}\}$ with the following frequencies:

Symbols	a	b	c	d	e	f	g
Frequencies	50	20	27	25	29	85	55

Also give the weighted length of the code (i.e, the sum over all symbols the frequency of the symbol times its encoding length).

Problem 2 We are given an array A of length n. For every integer i in $\{1, 2, 3, \cdots, n\}$, let b_i be median of the sub-array $A[1..i]$. (If the sub-array has even length, its the median is defined as the lower of the two medians. That is, if i is even, b_i is the $i/2$-th smallest number in $A[1..i]$.) The goal of the problem is to output $b_1, b_2, b_3, \cdots, b_n$ in $O(n \log n)$ time.

For example, if $n = 10$ and $A = (110, 80, 10, 30, 90, 100, 20, 40, 35, 70)$. Then $b_1 = 110, b_2 = 80, b_3 = 80, b_4 = 30, b_5 = 80, b_6 = 80, b_7 = 80, b_8 = 40, b_9 = 40, b_{10} = 40$.

Hint: use the heap data structure.

Problem 3 In the interval covering problem, we are given n intervals $(s_1, t_1], (s_2, t_2], \cdots, (s_n, t_n]$ such that $\bigcup_{i \in [n]} (s_i, t_i] = (0, T]$. The goal of the problem is to return a smallest-size set $S \subseteq \{1, 2, 3, \cdots, n\}$ such that $\bigcup_{i \in S} (s_i, t_i] = (0, T]$.

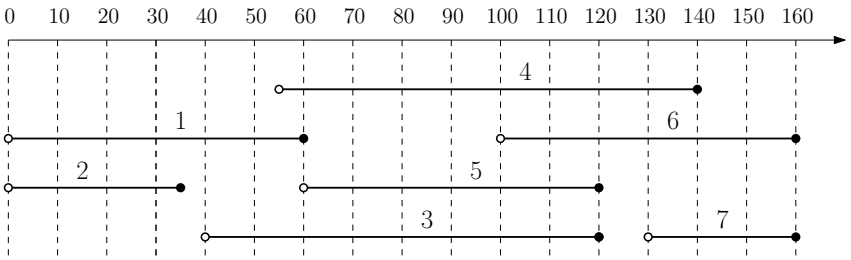
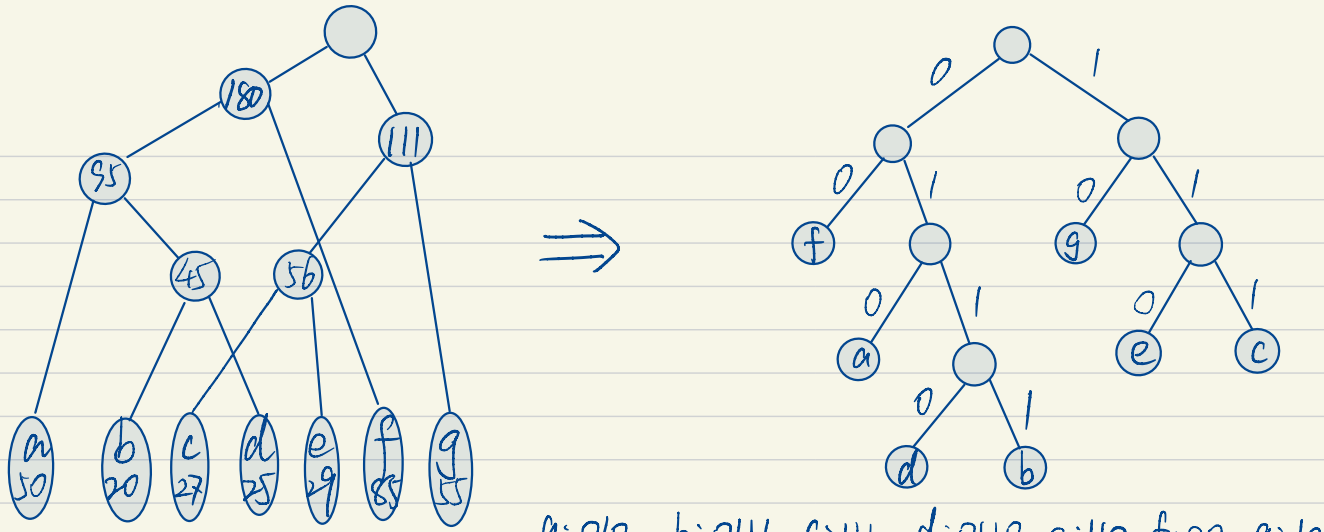


Figure 1: An instance of the interval covering problem.

Problem 1

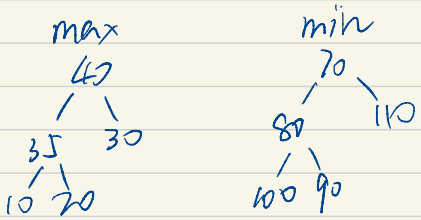


a: 010 b: 0111 c: 111 d: 0110 e: 110 f: 00 g: 10

Weighted length $W = 50\times3 + 20\times4 + 27\times3 + 25\times4 + 29\times3 + 85\times2 + 55\times2 = 778$

Problem 2

We use one max-heap for the $\lceil i/2 \rceil$ smallest numbers in $A[1...i]$, and a min-heap for the $\lceil i/2 \rceil$ largest numbers in $A[1...i]$. The odd and even lower medians are all coming from the maximum number of the max-heap.



Algorithm

```
1: maxh ← empty max-heap
2: minh ← empty min-heap
3: for i from 1 to n do
4:     if i is odd then
5:         if A[i] ≤ minh.getmin() or minh.isempty then
6:             maxh.add(A[i], A[i])
7:         else
8:             t ← minh.extractmin(), maxh.add(t,t), minh.add(A[i], A[i])
9:     else   /* i is even */
10:        if A[i] ≥ maxh.getmax() then
11:            minh.add(A[i], A[i])
12:        else
13:            t ← maxh.extractmax(), minh.add(t,t), maxh.add(A[i], A[i])
14:    b_i ← maxh.getmax()
15: return b
```

In each iteration for the loop at line 3, only the heap operations are peformed, so the running time for each iteration is $O(\log n)$. The loop has total of n times of iteration, so the total algorithm running time is $O(n \log n)$

For example, in the instance given by Figure 1. The intervals are $(0, 60], (0, 35], (40, 120], (55, 140], (60, 120], (100, 160], (130, 160]$. Then we can use 3 intervals indexed by 1, 4, 7 to cover the interval $(0, 160]$. This is optimum since no two intervals can cover $(0, 160]$.

Design a greedy algorithm to solve the problem. it suffices for your algorithm to run in polynomial time. To prove the correctness of the algorithm, it is recommended that you can follow the following steps:

- Design a simple strategy to make a decision for one step.
- Prove that the decision is safe.
- Describe the reduced instance after you made the decision.

Problem 3

1) let $c \leftarrow 0$
for all the interval i that $s_i \leq c$, include $(s_i, t_i]$ which has the largest t_i in the solution, and set $c \leftarrow t_i$
If multiple intervals has the same largest t_i, choose the one that has a larger s_i.

{ In simple words: each time pick the interval reaching furthest right, and also covering the end of previous interval.

2) Consider set S_c of all the subintervals that starts at or before S_c.
Since one of them must belong to the optimal solution, we replce it with a subinterval $(s_j, t_j]$ from S_c which the right endpoint t_j is maximal in S_c. In this case, the remain uncovered interval $(t_j, T]$ will be the subset of the remain interval from the optimal solution. So $(t_j, T]$ can also be covered by the same remain interval from the optimal solution.
That is to say, a solution built up with $(S_j, t_j]$ and the optimal solution for the remain interval $(t_j, T]$ will also be an optimal solution.

3) We remove all the intervals end at or before the end of previous included interval. Let c equal to the end of previous interval. The residual problem is defined by the set of remaining intervals.