# Midterm Solutions

# Problem 1

- We have two algorithms, A and B. Let $T_A(n)$ and $T_B(n)$ denote the time complexities of algorithm A and B respectively, with respect to the input size n.Complete the last column of the following table with A, B, or U, where:
- A means that for large enough n, A is always faster;
- B means that for large enough n, B is always faster;
- U means that the information provided is not enough to justify one algorithm is always faster than the other.

| | $T_A(n)$ | $T_B(n)$ | |
|---|---|---|---|
| 1 | $\Theta(n^{1.1})$ | $\Theta(n^{1.2})/(\log n)^3$ | A |
| 2 | $O(\log n)$ | $\Theta(\log \log n)$ | U |
| 3 | $O(n^2)$ | $O(n^{1.9} \log n)$ | U |
| 4 | $\Omega(n^2)$ | $O(n^{1.93})$ | B |
| 5 | $\Theta(n)$ | $\Theta(3^{\log_4 n})$ | B |

# Problem 2

Let the following comparison-based sorting algorithms:

- Insertion Sort (IS)
- Merge Sort (MS)
- Randomized Quick Sort (QS)
- Heap Sort (HS)

In the following questions, indicate each algorithm by its initials. There may be multiple algorithms for each question.

(1) Which of the above algorithms must be recursive?

MS and QS

(2) Which of the above algorithms is stable?

IS and MS

(3) Which of the above algorithms performs the minimum number of comparisons when the input array is already sorted?

IS

(4) Which of the above algorithms is optimal in terms of worst case running time?

MS and HS

(5) Which of the above algorithms uses the least amount of memory (working space)?

IS and HS

# Problem 3

Suppose we place n items randomly into m lockers, i.e. each locker has probability 1/m to contain a particular item. What is the expected number of pairs of items in the same locker?

Answer: This problem is the same as the birthday paradox, by viewing the n items as people and the m lockers as days. There are n(n-1)/2 distinct pairs of items, and the probability of each pair being in the same locker is 1/m. Thus, the expected number of pairs is:

$$\frac{n(n-1)}{2m}$$

# Problem 4

Given a min-heap in the array representation as follows,

3 14 7 31 15 8 10 34 32 17 18 9 12 19

(a) Show the new array after performing an extract-min operation.

7 14 8 31 15 9 10 34 32 17 18 19 12

(b) Show the new array after inserting 4 in the original array.

3 14 4 31 15 8 7 34 32 17 18 9 12 19 10

# Problem 5

You are given an array A of n integers in the range [1 .. k]. Describe a $\Theta(n + k)$ algorithm that generates a new array C, such that using C you can answer in constant time any query of the form "how many elements of A are in the range (x,y]" ($1 \leq x < y \leq k$).

Answer: You generate the same array C[1 .. k] as in counting sort, i.e. the element C[i] is the total number of elements smaller or equal to i. The answer to the query is: C[y]-C[x].

# Problem 6

Let A an unsorted array of n distinct numbers. Design an O(nlogk) algorithm that partitions A into k block of n/k numbers, such that all the numbers in block i are smaller than all the numbers in block i + 1, for 1 $\leq$ i < k. The numbers within each block do not have to be sorted. Assume that k << n, and that n, k are powers of 2.

Answer: Find the median of the array and partition it into two sub-arrays: O(n)

Recursively solve two subproblems until we have k blocks.

So T(n) = 2T(n/2) + cn. The base case is T(n/k) = 1. So, T(n) = O(nlogk).

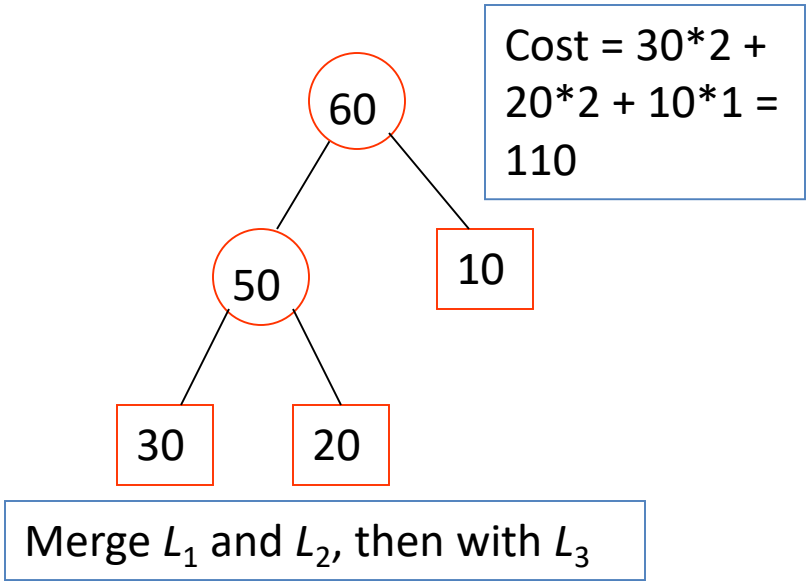
# Problem 7

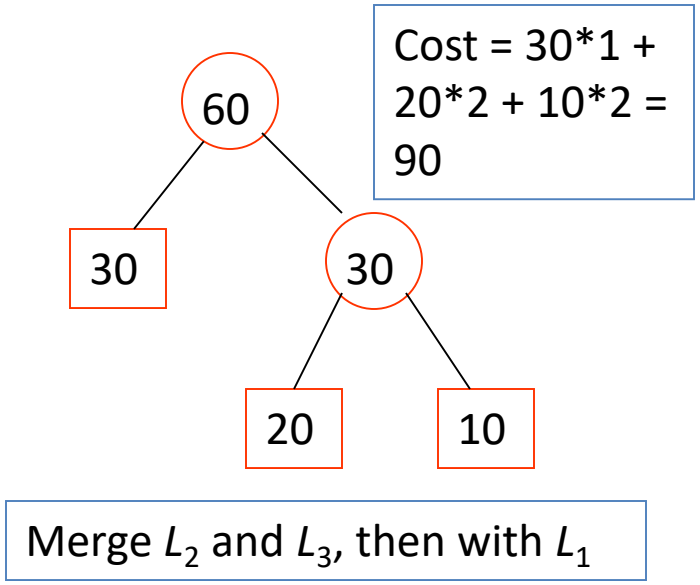
We are given n sorted lists $L_1$,.. ,$L_n$, which need to be merged into a single sorted list. We can merge only two lists at a time. Describe a greedy algorithm which determines the optimal order for merging all lists, i.e. the order that minimizes the total number of comparisons. Discuss the running time of your algorithm.

Clarification: The algorithm returns the optimal order, but it does not perform the actual merging. The lists have different number of elements.

# Binary Merge Tree

- A binary tree, built from the leaf nodes (the initial lists) towards the root in which each merge of two nodes creates a parent node whose size is the sum of the sizes of the two children.



# Optimal Binary Merge Tree Algorithm (same as Huffman)

- **Input:** $n \geq 2$ leaf nodes, each with a size (i.e., # list elements) .
- **Output:** a binary tree with the given leaf nodes which has a minimum total weighted external path lengths
- **Algorithm:**

Create a min-heap $T[1..n]$ based on the $n$ initial sizes.

While (the heap size $\geq 2$) do

delete from the heap two smallest values $a$ and $b$

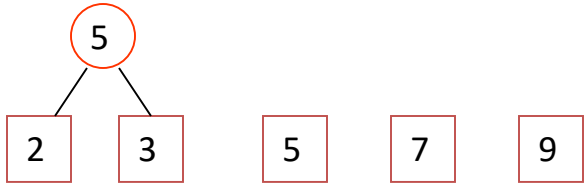
create a node of size $a + b$ whose children are $a$ and $b$

insert the value $(a + b)$ into the heap

- Time complexity $O(n \log n)$

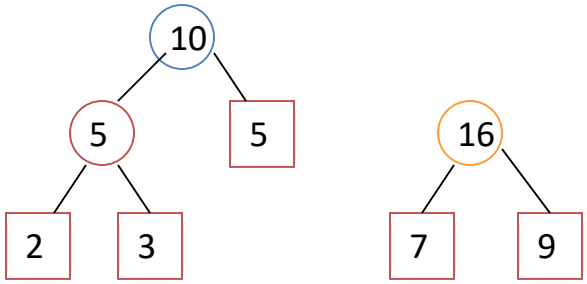
# Example of Optimal Merge Tree


Initially, 5 leaf nodes with sizes

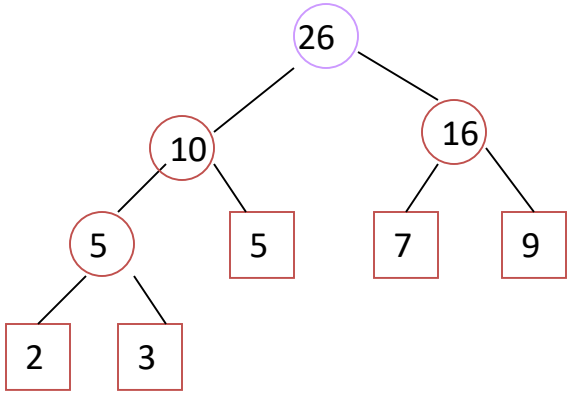



Iteration 1: merge 2 and 3 into 5

Iteration 2: merge 5 and 5 into 10




Iteration 3: merge 7 and 9 (chosen among 7, 9, and 10) into 16




Iteration 4: merge 10 and 16 into 26

Cost = 2*3 + 3*3 + 5*2 + 7*2 + 9*2 = 57.