Problems Related to Chapter 2 of Sedgewick-Wayne

Reiknirit, September 2025, v2

A) Illustrate / Match an Algorithm to Given Data

These problems ask you to trace, match, or identify algorithms based on intermediate results or invariants.

(*) Recommended problems

(#) Recommended if time

A1. (*) Heap Structure Checks

Given arrays, determine which are max-heaps, min-heaps, or neither:

- A1: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- A2: 9, 8, 5, 6, 7, 4, 3, 0, 1, 2
- A3: 0, 1, 3, 4, 5, 2, 6, 7, 8, 9

A2. (*) Partition Step of Quicksort (Pivot = First Element)

Apply one partitioning step of Quicksort (Sedgewick 2-partition, as in the book/slides) with pivot = 6 (the first element) to the array:

6, 3, 7, 9, 1, 8, 5, 2, 4

A3. Heap Insert

Start with the max-heap represented by the array:

16, 14, 10, 8, 7, 9, 3, 2, 4, 1

Insert the key 15. What is the resulting array representation of the heap?

A4. (#) Snapshots of Sorting (Large Array)

The array:

10, 15, 2, 3, 1, 4, 9, 0, 11, 8, 6, 12, 7, 14, 13, 5

is partially sorted. Match each result to its algorithm (SelectionSort, InsertionSort, MergeSort, bottom-up MergeSort, QuickSort):

- 6, 5, 2, 3, 1, 4, 9, 0, 7, 8, 10, 12, 11, 14, 13, 15
- 2, 3, 10, 15, 0, 1, 4, 9, 6, 8, 11, 12, 5, 7, 13, 14
- 0, 1, 2, 3, 4, 5, 6, 10, 11, 8, 9, 12, 7, 14, 13, 15
- 0, 1, 2, 3, 4, 9, 10, 8, 11, 15, 6, 12, 7, 14, 13, 5
- 0, 1, 2, 3, 4, 9, 10, 15, 6, 8, 11, 12, 7, 14, 13, 5

A5. Heap Delete-Max (delMax)

Consider the max-heap:

20, 15, 18, 8, 10, 12, 9, 6, 7, 2

After calling delMax, which of the following is the resulting heap array?

- 18, 15, 12, 8, 10, 2, 9, 6, 7
- 15, 10, 18, 8, 7, 12, 9, 6, 2
- 18, 15, 12, 8, 10, 9, 2, 6, 7

A7. (#) Trace the execution

For each algorithm we studied, trace the execution on the input array:

6, 2, 9, 5, 7, 0, 1

A8. Sequence of Heap Operations

We start with an empty max-heap. Perform the following sequence of operations:
insert(5), insert(12), insert(8), insert(20), insert(15), delMax(), insert(10)
What is the resulting heap array?

A9. (*) Heap Insertions and Properties

(a) Give the min-heap after inserting 5 into the heap: 0, 2, 1, 3, 6, 4, 8, 9, 7
(b) If you insert into a heap of size 7, which indices may change?
(c) In a max-heap with distinct elements, where can the largest, second-largest, and third-largest elements be?
(d) Is a sorted array always a min-heap?
(e) Give a linear-time algorithm to check whether an array satisfies the heap property.

A10. Snapshots of Sorting (Small Array)

We sort the array:

6, 2, 9, 5, 7, 0, 1, 8, 4, 3

with insertion sort, selection sort, merge sort, and quicksort.
Match each intermediate array snapshot to its algorithm:

- 1, 2, 3, 5, 4, 0, 6, 8, 7, 9
- 2, 5, 6, 7, 9, 0, 1, 8, 4, 3
- 0, 1, 2, 5, 7, 6, 9, 8, 4, 3
- 2, 6, 5, 9, 0, 7, 1, 8, 3, 4

B) Properties and Complexity

B1. (*) Complexity and properties

For each algorithm that we studied (Insertion,Selection,Merge,BU Merge, Quick,Heap):

- What is the worst-case and average case time complexity?
- Is it stable? in-place?
- When would it be the preferred choice?

B2. Comparisons vs Exchanges

A clerk must sort large crates. Comparisons are cheap, but exchanges are very costly, and there is only space to hold one extra crate.
Question: Which sorting algorithm should the clerk use and why?

B3. Three-Way MergeSort Complexity
Suppose MergeSort is modified to divide the input into three equal parts and merge them with a three-way merge.
Question: Why is the runtime still Θ(n log n)?

B4. (#) Sorting Invariants
(a) State the loop invariant maintained by insertion sort.
(b) State the loop invariant maintained by selection sort.
(c) How do they differ?

B5. (*) Consider an organ-pipe array that contains two copies of the integers 1 through N, first in ascending order, then in descending order. For example, here is the array when $N = 8$:

1 2 3 4 5 6 7 8 8 7 6 5 4 3 2 1

Note that the length of the array is $2N$, not N.
(a) How many compares does selection sort make to sort the array as a function of N? Use tilde notation to simplify your answer.
$\sim$ compares

(b) How many compares does insertion sort make to sort the array as a function of N? Use tilde notation to simplify your answer.
$\sim$ compares

(c) How many compares does merge sort (the basic top-down version, without any practical improvements) make to sort the array as a function of N? Assume N is a power of 2. Use tilde notation to simplify your answer.
$\sim$ compares

B6. 4-way Heaps
How many comparisons are used when inserting into a 4-way heap? How does it compare with a binary heap?

C) Applications
Unless otherwise stated, give an efficient algorithm to solve the problem in general. It should run in $O(n)$ or $O(n \log n)$, where n is the length of the input (usually array). **Do not use hashing.**

C1. Intersection of Arrays
Given two arrays, for example:
A = 3, 5, 7, 9, 11
B = 2, 3, 6, 9, 12
determine which elements appear in both.

C2. (*) Duplicates Check
Given an array, for example:

4, 1, 7, 2, 7, 3, 9
determine if all elements are distinct.

C3. Most Frequent Element
Given array, find the most frequent element.
Example:
A = 3, 7, 3, 2, 5, 2, 3, 5, 5, 5

C4. (*) Bitonic Array Sorting
Explain how to sort a bitonic array of length n in linear time.
Example: 2, 5, 8, 10, 7, 4, 1

C5. k-th Smallest Element
Give a $\Theta(n \log n)$ algorithm to find the k-th smallest element in an array.
Example: Array = 9, 4, 7, 11, 5, 8, 1, 6, 3, k = 4, should output 5.

D) Problem Solving (Design an Algorithm)

(These ask the student to propose or analyze an algorithm to solve a new problem.)

Formulate the problem computationally (stating the input and the output). Then give an efficient algorithm to solve the problem in general. **Do not use hashing.** It should run in $O(n)$ or $O(n \log n)$ time, where n is the length of the input. Explain briefly why it solves the problem and why the running time holds.

D1. Election Winner
Given votes as candidate names, find the winner efficiently.
Example:
Votes = A, B, A, C, B, A, B, B
Optional: If candidates are numbers 1–10, does this change?

D2. (Moved to E)

D3. Two-Sum Problem
Given an array a and integer k, find if there exist distinct indices i, j such that $a[i] + a[j] = k$.
For example, on a = 4, 2, 7, 1, 5, 9 and k = 12 the answer is yes (a[2]+a[5]=5+7=k).

D4. (Now C5)

D5. Median Data Structure
Design a data structure that supports insertion in $O(\log n)$, median query in $O(1)$, and median

deletion in $O(\log n)$. (Note that the only element we want access to is the median, but do not need access to any other element.)

D6. (See D11).

D7. (#) k Sorted Arrays Merge

Given k sorted arrays totaling N elements ($k < N$), output all elements in sorted order in $O(N \log k)$ time.

Example: k = 3, N=9, Array 1 = 1, 5, 9, Array 2 = 2, 4, 7, Array 3 = 3, 6, 8

D8. (See D3)

D9. Median of Two Sorted Arrays

You are given two sorted arrays of total size n. Design an $O(\log n)$ time algorithm to compute the median.

Example: Array 1 = 1, 3, 8, Array 2 = 2, 5, 7, 9. Median is 5.

D10. Dynamic Top-k Elements

Design a data structure that (given in advance a value k) supports:

- inserting an element in $O(\log k)$,
- reporting the k largest elements seen so far in $O(k)$.
  Example stream (k = 3): 5, 12, 8, 20, 15, 10, the k largest elements are 12, 20, 15.

D11. **(*)** Jón wants to buy a house on a suburban street. There are *n* identical houses and they are all available for sale. However, houses are not evenly spaced on the street and Jón wants to be as far a possible from its closest neighbors. Given the list of positions of every house on the street, design a Θ(n log n) time algorithm to find the house that suits Jón the best.

For instance, if houses are placed at positions: 4, 3, 1, 7, 11 and 10, then Jon will prefer the one placed at position 7 since its closest neighbors are at distance 3 while every other house has a neighbor within distance 2.

D12. **(*)** A reporter wants to find out what point difference occurs the most frequently in games in the football league. Namely, if one team scores 68 and the other 75, the point difference is 7. He has the outcome of all n games in the league (stored in an array) and needs your help determining the most common point difference. Describe an Θ(n log n) algorithm to do so.

For instance, if the games went: 75 vs. 67, 95 vs. 80, 92 vs. 100, then 8 is the most common point difference.

D13. (Deleted)

E) Extensions

These are related problems that are slightly outside the scope of the course.

E1. Sorting Immovable Elements

We must sort N elements that are too large to move. How can this be done?

E2. External Sorting

You need to sort 10 GB of data, but your memory can hold only 500 MB at a time. Describe an efficient external sorting strategy.

E3. Ranking Agreement (Ken & Barbie)

Ken and Barbie rate films. Their "fit score" counts pairs of films ordered consistently in both preference lists.

Example:

Ken = A, B, C, D

Barbie = B, A, D, C

Compute the number of consistent pairs efficiently.

E4. Dutch National Flag Problem

(a) Sort an array containing only 0, 1, 2 in linear time using only 3 variables.

Example: 2, 0, 1, 2, 0, 1, 1, 0

(b) If the array contains values 0, 1, ..., k, how fast can you sort it as a function of n and k? How much space?

E5.

The following function solves the problem Continuous Median on Kattis. It returns the sum of the medians of all prefixes of the array a[].

```
static long solve(int[] a) {
    long sum = 0, med;
    int n = a.length;
    for (int j = 1; j < n; j++) {
        Arrays.sort(a, 0, j); // sort subarray a[0..j)
        int middle = (j / 2);
        if (j % 2 == 0) {
            med = (a[middle] + a[middle + 1]) / 2;
        } else {
            med = a[middle];
        }
        sum += med;
    }
    return sum;
```

}

A. What is the order-of growth time complexity of the function as a function of n?

B. How can we rewrite the function so that the code ends before the 1 second time limit on instances of size $n = 10^5$? You may describe it in code, pseudocode, or in words.

E6. Restricted Deck Sorting

Sort a deck using only the operations: look at the top two cards, exchange them, or move the top card to the bottom.

Example initial deck: 4, 1, 3, 2