AY1819 S1 Midsem Test

Selected Answers

Qn A

1. O(1), size of stored by std::vector
2. O(N), requires looping through the entire string
3. O(N), “insertion sort”, PS1B
4. $O(N^2)$, worse-case for non-randomized quick sort is sorted array
5. O(N log N), just substitute N = N/2 and simplify
6. O(N), iterate through the linked list
7. O(1), have head and tail pointers
8. O(N), need to pop N-1 times
9. O(N), same as reversing Linked List/Stack
10. O(1), pop_back and pop_front are both O(1)

Qn B1

When writing a C++ class, it is possible to not implement/define any constructors and still be able to compile the program.

True. --1m

“I used it before” -- 2m

“A default constructor …” -- 3m

Qn B2

There is no sorting algorithm that is both in-place and stable.

False.

Refer to table in lecture notes!

	Worst Case	Best Case	In-place?	Stable?
Selection Sort	$O(n^2)$	$O(n^2)$	Yes	No
Insertion Sort	$O(n^2)$	$O(n)$	Yes	Yes
Bubble Sort	$O(n^2)$	$O(n^2)$	Yes	Yes
Bubble Sort 2	$O(n^2)$	$O(n)$	Yes	Yes
Mergesort	$O(n \lg n)$	$O(n \lg n)$	No	Yes
Radix sort	$O(dn)$	$O(dn)$	No	yes
Quicksort	$O(n^2)$	$O(n \lg n)$	Yes	No

Qn B3

Merge sort is **O(N)** if we divide it into **N** parts?

False.

The analysis assumes that we can merge all N parts and still takes O(**N**) time.

In reality, we can only merge 2 parts in O(**N**) time. Merging **N** parts will take **O(N log N).**

Qn B4

Insertion sort will *never* run faster than Mergesort when sorting the same array.

False.

For a sorted array, insertion sort is O(N) while mergesort will still take O(N log N).

Qn B5

Can we print SLL in reverse order in O(N)?

True.

Copy to stack and print -- O(N)

Reverse the SLL, print, reverse back. -- O(N)

Question **did not** mention memory usage, just asking if it is possible. Any memory usage is ok.

Qn C1 - Best Case of Non-Rand Quick Sort

For the best case, we want the partition to split into 2 “equal” parts.

When is that the case for the first partition?

→ When the pivot is the **median**.

Qn C1 - Best Case of Non-Rand Quick Sort

First number: 4

Left segment: [1, 2, 3], right segment: [5, 6, 7]

Number of comparisons: 6

Solve [1, 2, 3] and [5, 6, 7] separately.

Qn C1 - Best Case of Non-Rand Quick Sort

For [1, 2, 3]:

Median: 2

Partition into [1] and [3] using **2** comparisons.

For [5, 6, 7]:

Median: 6

Partition into [5] and [7] using **2** comparisons.

Qn C1 - Best Case of Non-Rand Quick Sort

Putting together

[2, 1, 3], 4, [6, 5, 7] (Ideal case after 1st partition)

4, 1, 3, **2**, 6, 5, 7

(Last step of partition swaps the pivot into the correct position)

Other variants accepted.

Qn C1 - Best Case of Non-Rand Quick Sort

Tentative Marking scheme:

10 comparisons -- 5 marks

11 or 12 comparisons -- 4 marks

> 12 comparisons -- 0

Qn C1 - Best Case of Non-Rand Quick Sort

```
void partition(int A[], int s, int e) {
    if (s+1 >= e) return;
    int pivot = A[s];
    int k = s;
    for (int i = s+1; i < e; ++i) {
        ++num_comparisons;
        if (A[i] < pivot) {
            ++k;
            swap(A[i], A[k]);
        }
    }
    swap(A[s], A[k]);
    partition(A, s, k);
    partition(A, k+1, e);
}
```

Qn C2

Make selection sort slower than radix sort.

Selection sort: $O(N^2)$

Radix sort: $O(dN)$, where $d = 3$.

Big-O is asymptotic analysis.

Qn C2

“Just make **N** very large”

Safest: >= 10 integers

We give 5m for >= 7 integers.

Partial for between 4 to 7.

N = 3 is not accepted.

Qn C3

Stack: First In Last Out → Flipped order

Queue: First In First Out → Same order

As long as input array is **palindromic**.

X = [1, 2, 3, 4, 5, 4, 3, 2, 1]

Reads the same from front and back.

Qn D1 - Special Case 1 (K = N)

All the students must be selected into the project.

→ There is only 1 possible group, everyone.

Find the max - min.

Each can be done in **O(N)** time.

Qn D1 - Special Case 2 (K = 2)

Only 2 students are to be selected.

$O(N^2)$ approach:

Try every pair of 2 students, print the minimum difference.

Qn D1 - Special Case 2 (K = 2)

Observation

Lets say you **must** include student X with score 50.

Since you want to minimize the difference, you either pair him with a student that is “closest to 50”.

Qn D1 - Special Case 2 (K = 2)

Observation

That means you can check the next *highest* or next *lowest* scoring student.

If you sort the list of scores, these 2 students are **adjacent to X!**

Qn D1 - Special Case 2 (K = 2)

Approach

Sort the list of students, output the minimum difference between every 2 adjacent pair of sorted students scores.

O(N log N)

Qn D1 - General Case

Observation

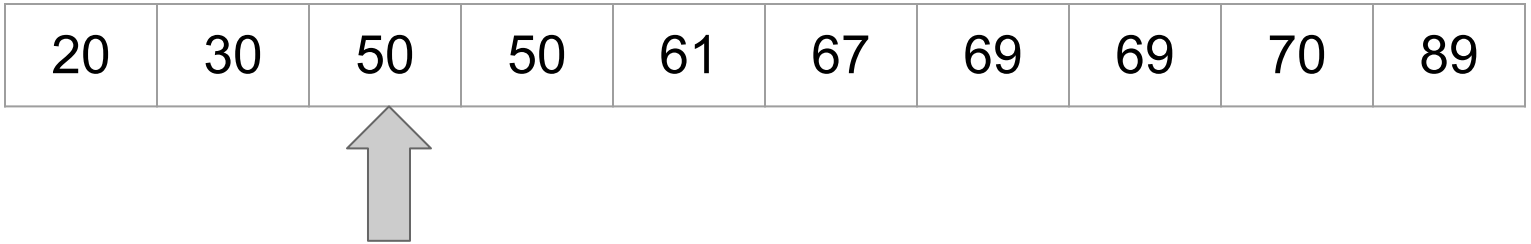
Lets say you **must** include student X with score 50 and **he/she is the lowest scoring student**.

You will need to select **K** students that scores closest to 50, but >= 50.

Qn D1 - General Case

Observation

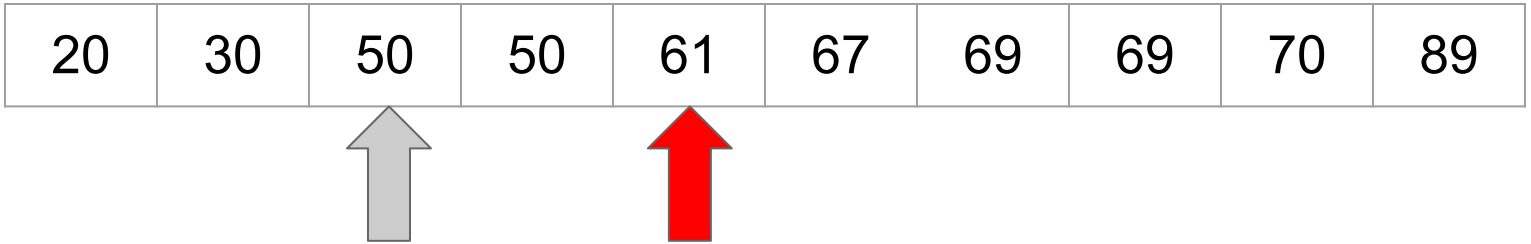
When K = 3, i = 2



Qn D1 - General Case

Observation

When K = 3, i = 2

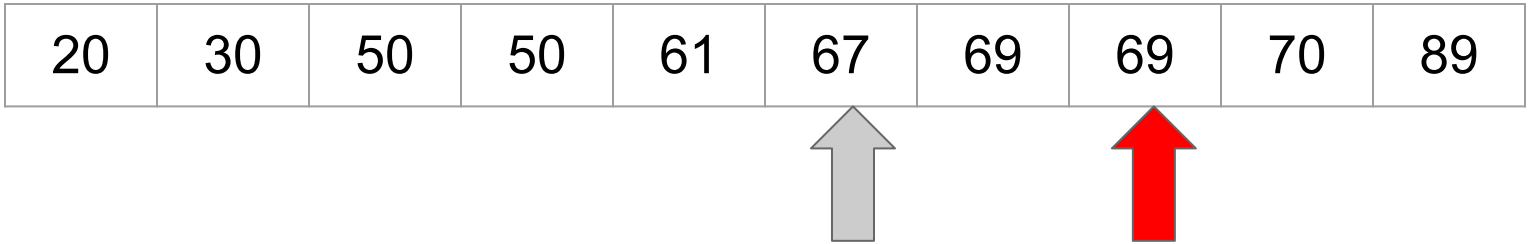


Take A[i+K-1] - A[i]

Qn D1 - General Case

Approach

We can calculate this for every value of i.



O(N) for this loop

Qn D1 - General Case

Approach

Total time complexity: O(N log N) from STL sort

```
sort(scores, scores+N);          //O(N log N) time complexity
int ans = scores[N-1] - scores[0];          //maximum possible answer
for (int i = 0; i + K - 1 < N; i++) {
      //Attempt to group students i to i + K - i
      //Since they are sorted, the difference is just scores[i+K-1] - scores[i]
  ans = min(ans, scores[i + K - 1] - scores[i]);
}
cout << ans << endl;
```

Qn D2 - Quick Question

Which LDS does the bus **most resemble**?

Stack. (or STL Stack, Stack ADT)

"*First-In-Last-Out*"

Linked List and Deque not Accepted as the question asks for **"most"** resemble.

Qn D2 - General Case

Short Answer: Bracket Matching

Each person is a unique pair of ‘brackets’.

At the boarding station: Open Bracket

At the alighting station: Closed Bracket

Qn D2 - General Case

If the sequence is valid: then it is possible for the bus to transport everyone.

Otherwise, it is not!

Qn D2 - General Case

Sample Input 1:

Alight					P4				P2	P3		P1
#	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Board	P1	P4			P3	P2						
Brackets	**(**	**<**			**>{**	**[**			**]**	**}**		**)**

Qn D2 - General Case

Sample Input 2:

Alight			P3				P4		P1	P2	
#	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Board	P1	P2	P4		P3						
Brackets	**(**	**<**	**} [**		**{**		**]**		**)**	**>**	

Qn D2 - General Case

Some edge cases:

- A person alight and board at the same stop.
  - $S_i == E_i$
- Need to sequence alights before boards.

Will not be heavily penalized for these.

Qn D2 - General Case

Approach

- Construct the ‘brackets’
- Can cite “bracket matching algorithm”

Any *direct simulation* in O(N) will also be accepted.