CSc 345 Project 2 Report

Qiwei Li

TOTAL POINTS

23 / 27

QUESTION 1

1 Part 1 Questions 7 / 9

- **0 pts** Correct
- **1 pts** input
- ✓ **- 1 pts** **All parts**
- ✓ **- 1 pts** **Compare**
- **1 pts** Best Case
- **1 pts** runtime
- **0.5 pts** Queue
- **1 pts** Worst case
- **3.5 pts** 6 runtimes
- **7 pts** No answer
- **9 pts** None

Resizing takes O(N)

QUESTION 2

2 Part 2 Questions 7.5 / 9

- **0 pts** Correct
- ✓ **- 1.5 pts** **A little more**
- **1.5 pts** Recurrence relation
- **1 pts** Description
- **1.5 pts** Worst Case
- **0.5 pts** Off a little
- **1 pts** runtime
- **1.5 pts** Best case
- **0.5 pts** big oh notation
- **9 pts** none
- **0.5 pts** reason
- **3 pts** Comparison

A little more on the comparison and on how you got to the recursive relations.

QUESTION 3

3 Part 3 Questions 8.5 / 9

- **0 pts** Correct
- ✓ **- 0.5 pts** **runtime**
- **1 pts** Discuss
- **2 pts** 3 runtimes
- **1 pts** 2 runtimes

Question 1

(1)

The first element must be the starting index of first subarray. For other starting indices, we need to find all i that meet a[i]<a[i-1]. The indices i are the left starting indices. Given the starting indices, the ending indices are easy to obtain. The runtime of portion is O(n).

(2)

I use a queue to store the starting and ending indices of all subarrays. Every time, pop 2 subarrays and merge them, and push the new starting and ending indices of the merged subarray to the queue. If we use a stack instead of queue, the process is that we always merge the first 2 subarrays and put the new subarray to be the new top of stack. which is slower than using queue.

(3)

The time of SortMerge is related to the number of sorted subarrays. Suppose the number of sorted subarrays is K, the running time is n(K/2+K/4+⋯+1) = nlogK. When K=n, it is the worst case and the time is O(nlogn). When K=1, it is best case and the time is O(n). The worst case happens when the array is in descending order. The best case happens when the array is in ascending order.

(4)

This algorithm is better than regular mergesort. The running time of regular mergesort is always O(nlogn), which is worse than SortMerge when the array is in ascending order.

Question 2

(1)

Under the best case, the recurrence relation is T(n)=3T(n/3)+O(n), so T(n)=$O(nlog_3 n)$. When the two pivots always divide the array into 3 almost equal subarrays in each iteration, it reaches the best case.

Under the worst case, the recurrence relation is T(n)=T(n-2)+O(n), so T(n) = O(n^2). When the 2 pivots are always in the two endpoints of the array to sort, for example, the array is already sorted, it reaches the worst case, because it only reduces the array to sort by 2 elements.

(2)

I think the Five-way Quicksort is better, because the best time is $O(nlog_3 n)$,which is better than $O(nlog_2 n)$ achieved by Three-way Quicksort. However, the constant of Five-way Quicksort is bigger, which may influence the running time in small data size.

Question 3

(1)

We could not find the position of the element to be updated or removed with the max-heap. Without the hashtable, we have to go through the heap to find the target element, and it need O(n) time. In order to find the position of target element in O(1) time, we need a

1 Part 1 Questions 7 / 9

- **0 pts** Correct
- **1 pts** input
- ✓ **- 1 pts** **All parts**
- ✓ **- 1 pts** **Compare**
- **1 pts** Best Case
- **1 pts** runtime
- **0.5 pts** Queue
- **1 pts** Worst case

Question 1

(1)

The first element must be the starting index of first subarray. For other starting indices, we need to find all i that meet a[i]<a[i-1]. The indices i are the left starting indices. Given the starting indices, the ending indices are easy to obtain. The runtime of portion is O(n).

(2)

I use a queue to store the starting and ending indices of all subarrays. Every time, pop 2 subarrays and merge them, and push the new starting and ending indices of the merged subarray to the queue. If we use a stack instead of queue, the process is that we always merge the first 2 subarrays and put the new subarray to be the new top of stack. which is slower than using queue.

(3)

The time of SortMerge is related to the number of sorted subarrays. Suppose the number of sorted subarrays is K, the running time is n(K/2+K/4+⋯+1) = nlogK. When K=n, it is the worst case and the time is O(nlogn). When K=1, it is best case and the time is O(n). The worst case happens when the array is in descending order. The best case happens when the array is in ascending order.

(4)

This algorithm is better than regular mergesort. The running time of regular mergesort is always O(nlogn), which is worse than SortMerge when the array is in ascending order.

Question 2

(1)

Under the best case, the recurrence relation is T(n)=3T(n/3)+O(n), so T(n)=$O(nlog_3 n)$. When the two pivots always divide the array into 3 almost equal subarrays in each iteration, it reaches the best case.

Under the worst case, the recurrence relation is T(n)=T(n-2)+O(n), so T(n) = O(n^2). When the 2 pivots are always in the two endpoints of the array to sort, for example, the array is already sorted, it reaches the worst case, because it only reduces the array to sort by 2 elements.

(2)

I think the Five-way Quicksort is better, because the best time is $O(nlog_3 n)$,which is better than $O(nlog_2 n)$ achieved by Three-way Quicksort. However, the constant of Five-way Quicksort is bigger, which may influence the running time in small data size.

Question 3

(1)

We could not find the position of the element to be updated or removed with the max-heap. Without the hashtable, we have to go through the heap to find the target element, and it need O(n) time. In order to find the position of target element in O(1) time, we need a

2 Part 2 Questions 7.5 / 9

- **0 pts** Correct

✓ - **1.5 pts** **A little more**

- **1.5 pts** Recurrence relation
- **1 pts** Description
- **1.5 pts** Worst Case
- **0.5 pts** Off a little
- **1 pts** runtime
- **1.5 pts** Best case
- **0.5 pts** big oh notation
- **9 pts** none
- **0.5 pts** reason
- **3 pts** Comparison

A little more on the comparison and on how you got to the recursive relations.

hashtable to map the patient name to the patient information. As we know, the time of swim/sink/delete an element in heap is O(logn). Once we could locate the target in O(1) time, we just need to spend O(logn) time to adjust the heap.↵

↵

(2)↵

- process: The best runtime is O(1), when the patient was sent to the ER. The worst runtime is O(logN), when the patient was inserted into the queue. The expected runtime is O(logN).↵
- seeNext: The best/worst/expected runtime is O(logN). No matter in which case, we need to delete the Max in queue that cost O(logN) time.↵
- handle_emergency: The best/worst/expected runtime is O(logN). The time of update function and remove function are both O(logN). The best case is only the update is called and the runtime is O(logN). The worst case is the remove and update fucntions are called and the runtime is O(logN+logN)=O(logN). So, the expected time is O(logN).↵
- walk_out: The best/worst/expected runtime is O(logN). The time of remove function is O(logN) and we always and only call the remove function once.↵

3 Part 3 Questions 8.5 / 9

- **0 pts** Correct

✓ **- 0.5 pts** **runtime**

- **1 pts** Discuss
- **2 pts** 3 runtimes
- **1 pts** 2 runtimes
- **3.5 pts** 6 runtimes
- **7 pts** No answer
- **9 pts** None

Resizing takes O(N)