CISC 2200 - Data Structures Homework 6 Dr. Rahouti

Fall, 2022 100 points

Exercise 1 (15 pts—5 pts/Q):

Consider the following array:

41	7	11	22	17	3	19	5	58	13
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Show the content of the array after the fourth iteration of

a. Bubble Sort
b. Selection Sort
c. Insertion Sort

Exercise 2 (10 pts):

Consider the following array (same as in Exercise 1):

41	7	11	22	17	3	19	5	58	13
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Show how the values in the array would have to be rearranged in order to satisfy the heap property.

Exercise 3 (15 pts—5 pts/Q):

Consider the following array:

26	24	3	17	25	24	13	60	47	1
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Tell which sorting algorithm would produce the following results after four iterations:

a.

1	3	13	17	26	24	24	25	47	60
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

b.

1	3	13	17	25	24	24	60	47	26
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

c.

3	17	24	26	25	24	13	60	47	1
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Exercise 4 (5 pts):

How many comparisons would be needed to sort an array containing 100 elements using Selection Sort if the original array values were already sorted?

Exercise 5 (10 pts):

How would you modify the Radix Sort algorithm to sort the list in descending order than ascending order?

Exercise 6 (10 pts):

Determine the Big-O measure for QuickSort based on the number of elements moved rather than the number of comparisons.

a. For the best case
b. For the worst case

Exercise 7 (10 pts):

Does Radix Sort return correct sorting results when the input sequence contains negative elements? If yes, please give your reason. If no, please revise LSD Radix Sort algorithm to deal with negative elements and give your pseudocode.

Exercise 8 (10 pts):

Suppose ***n*** is even and array $A = [a_1, a_2, ..., a_n]$ is semi-identical, i.e., $a_1 = a_2 = ... = a_n/2$ and $(a_n/2)+1 = (a_n/2)+2 = ... = a_n$. For example, $A = [1, 1, 1, 1, 2, 2, 2, 2]$. What is the quicksort's running time in this case? Please explain your reason.

Exercise 9 (5 pts):

Use the Three-Question Method to verify MergeSort algorithm.

Exercise 10 (10 pts):

The *C++ thread* library provides a function that returns the number of threads that the hardware is capable of running. Modify the parallel *MergeSort* so that the user specifies a minimum chunk size. The program should then use the number of threads available and *MAX_ITEMS* to determine the largest chunk size that will produce that many threads. If the computed chunk size is larger than what the user specifies, use that instead. Here is how to get the number of threads available:

unsigned int maxthreads = thread::hardware_concurrency();

Consider the following array:

41	7	11	22	17	3	19	5	58	13
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Show the content of the array after the fourth iteration of

a. Bubble Sort
b. Selection Sort
c. Insertion Sort

a. 1st iteration

3 41 7 11 22 17 5 19 13 58

2nd iteration.

3 5 41 7 11 22 17 13 19 58

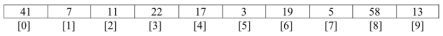
3r iteration

3 5 7 41 11 13 22 17 19 58

4th iteration

3 5 7 11 41 13 17 22 19 58

b For selection sort



41	7	11	22	17	3	19	5	58	13
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

1st:

3 7 11 22 17 41 19 5 58 13

2nd:

3 5 11 22 17 41 19 7 58 13

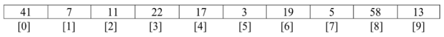
3rd:

3 5 7 22 17 41 19 11 58 13

4th:

3 5 7 11 17 41 19 22 58 13

c. insertion sort



41	7	11	22	17	3	19	5	58	13
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

1st:

41 7 11 22 17 3 19 5 58 13

2nd.

7 41 11 22 17 3 19 5 58 13

3rd:

7 11 41 22 17 3 19 5 58 13

4th:

7 11 22 41 17 3 19 5 58 13

Exercise 2 (10 pts):

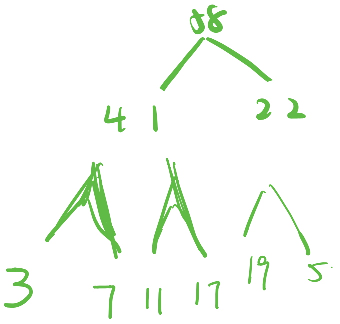
Consider the following array (same as in Exercise 1):

41	7	11	22	17	3	19	5	58	13
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Show how the values in the array would have to be rearranged in order to satisfy the heap property.

assume we refer to max-heap property. [parent > left && prt > right

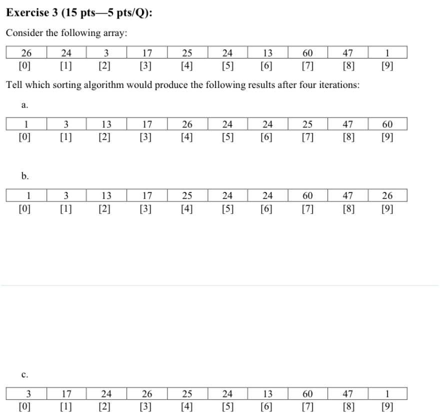
First, we draw a tree that satisfies the max-heap property.



∴ the array can be

58 41 22 3 7 11 17 19 5.

note: for min-heap, the array is different, but logic is same.



Exercise 3 (15 pts—5 pts/Q):

Consider the following array:

26	24	3	17	25	24	13	60	47	1
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Tell which sorting algorithm would produce the following results after four iterations:

a.

1	3	13	17	26	24	24	25	47	60
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

b.

1	3	13	17	25	24	24	60	47	26
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

c.

3	17	24	26	25	24	13	60	47	1
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

a Bubble sort

b selection sort

C. insertion sort.

Exercise 4 (5 pts):

How many comparisons would be needed to sort an array containing 100 elements using Selection Sort if the original array values were already sorted?

If the array is sorted, selection sort is O(n)

∴ There are 99 comparisons [no need to compare with itself]

Exercise 5 (10 pts):

How would you modify the Radix Sort algorithm to sort the list in descending order than ascending order?

By reversing the order of digit in each step of the algorithm

The pseudocode:

```
RadixSort(A)
    max_digit = max( numberOfDigits(A[i]), i=1 to length(A) )
    for j = max_digit down to the number 1   // reverse iterate.
        stable_sort(A, j)   // use stable_sort. to sort the elements
```

Exercise 6 (10 pts):

Determine the Big-O measure for QuickSort based on the number of elements moved rather than the number of comparisons.

a. For the best case
b. For the worst case

a. $O(n \log n)$

b. $O(n^2)$

Exercise 7 (10 pts):

Does Radix Sort return correct sorting results when the input sequence contains negative elements? If yes, please give your reason. If no, please revise LSD Radix Sort algorithm to deal with negative elements and give your pseudocode.

No, it can't. we have to use the absolute vale to the negative numbers, then sort it using Radix sort as in the slides. lastly, move the negative numbers to the front, presere the relative orders.

```
LSD-Rad_Sort ( A )
    for i=1 to length (A)
        A[i] = abs (A[i])      // convert to abs first.

    for d=1 to max.Num Of Digits
        stable Sort (A, d )

    negative_ones = [ ]        // new array to store the negative values

    for i=1 to length (A )
        if A[i] <0
            negative_ones . append ( A[i] )

        else
            break.

    A= concat ( negative_ones, A[ i+1 : length (A) ] )    // negative + non negative
                                                          // new sorted array
```

Exercise 8 (10 pts):

Suppose $\boldsymbol{n}$ is even and array $A = [a_1, a_2, \ldots, a_n]$ is semi-identical, i.e., $a_1 = a_2 = \ldots = a_n/2$ and $(a_n/2)+1 = (a_n/2)+2 = \ldots = a_n$. For example, $A = [1, 1, 1, 1, 2, 2, 2, 2]$. What is the quicksort's running time in this case? Please explain your reason.

it also depends. if the pivot point is in the middle, it will partition half of the arrays the running time is $O(n \log n)$. If chosen from the endpoint. it will be $O(n^2)$

Exercise 9 (5 pts):

Base. case. recursion.
general.
does it solve small instance of smaller

Use the Three-Question Method to verify MergeSort algorithm.

Based on the professor's mentioned in class, this question should be answered in base case, general case method.

Base case: when the array is sorted in only one element in the subarray.

general case: when there aren't one element [>1]

First, divide the arrays into two subarrays, till one subarray has one element. [Divide]

then we merge these 1 elements array while sorting till it forms a sorted array w/ same amount of elements w/ our original arrays.

[Conquer]

Exercise 10 (10 pts):

The *C++ thread* library provides a function that returns the number of threads that the hardware is capable of running. Modify the parallel *MergeSort* so that the user specifies a minimum chunk size. The program should then use the number of threads available and *MAX_ITEMS* to determine the largest chunk size that will produce that many threads. If the computed chunk size is larger than what the user specifies, use that instead. Here is how to get the number of threads available:

unsigned int maxthreads = thread::hardware_concurrency();