Data Structure ; CS 21200
Fall 2020
Prof George Wolberg

Md Shahid Ferdaus
23726181

Final (Chap 10-13)

Problem 10.1

B

Problem 10.2

B

Problem 10.3

B

Problem 10.4

D

Problem 10.5

B

Problem 10.6

A

Problem 10.7

A

Problem 10.8

E

Problem 10.9

C

Problem 10.10

C

Problem 10.11

B

Problem 10.12

A

Problem 10.13

D

Problem 10.14

D

Problem 10.15

B

Problem 10.16

C

Chapter 11

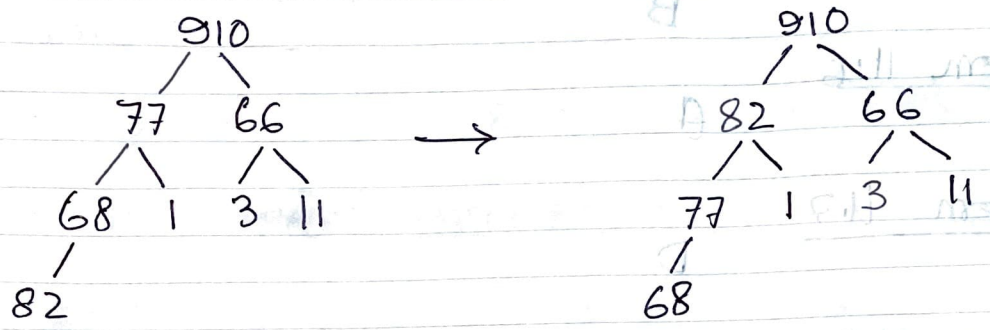
Problem 11.1

2 reasons:

1. The parent has to be equal or bigger than the 2 children. [3 is bigger than 2]

2. Tree is not a complete tree.

Problem 11.2



Problem 11.3

If you are a leaf, you can not go down anymore. The reheapification downward process would also stop when the current node is not greater than or equal to its both child or not.

Problem 11.4

If you are greater than your parents, then you can go up. The condition for reheapification upward is in each step to check whether the key value of current node is ~~less~~ greater than its parent or not. If not, we stop reheapification upward.

Problem 11.5

B

Problem 11.6

A

Problem 11.7

D

Problem 11.8

A

Problem 11.9

B

Problem 11.10

D

Problem 11.11

C

Problem 11.12

A

Problem 11.13

A

Chapter 12

Problem 12.1

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

→ From the ~~beginning~~ begining:

(1) (2) 3 4 5 6 7 8 9 10 11 12 13 14 15

→ From the ending:

1 2 3 4 5 6 7 8 9 10 11 12 13 (14) (15)

Problem 12.2

1 2 3 (4) 5 6 7 (8) 9 10 11 (12) 13 14 15

Problem 12.3

C

Problem 12.4

B

Problem 12.5

C

Problem 12.6

C

Problem 12.7

E

Problem 12.8

B

Chapter 13

Problem 13.1

5 3 8 9 1 7 0 2 6 4

first iteration: 0 3 8 9 1 7 5 2 6 4

Problem 13.2

5 3 8 9 17 0 2 6 4

1st iteration: 3 5 8 9 17 0 2 6 4

Problem 13.3

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

Ans. 4, 5 and 9.

Problem 13.4

A good way for quicksort to choose a pivot element is perform partition function. The partition function chooses some arbitary pivot elements and places it at the correct index positions. and finally divides the remaining array elements. However, all three steps does not necessarily happens in that order, but it completes the 3 steps.

Problem 13.5

5 3 8 9 1 7 0 2 6 4

using 5 pivot → ~~3 5 8 9 1 7 0 2 6 4~~

4 3 1 0 2 5 9 8 6 7

Problem 13.6

5 3 8 9 1 7 0 2 6 4

merge sort → 1 3 5 8 9 0 2 4 6 7

Problem 13.7

Problem 13.7

```
void merge(int data[], size_t n1, size_t n2);
{int *temp; // points to dynamic array to hold
              elements.

size_t copy1, copy2, copy3 = 0;

// copy1, copies the number of elements from data to temp
// copy2, copies 1st half of data
// copy3, copies 2nd half of data.

size_t ~~copy_back~~; // to copy back from temp to data.

// making memory for the temp. dynamic array

temp = new int [n1+n2];
// merging elements
while((copy2 < n1) && (copy3 < n2)){
    if (data[copy2] < (data+n1)[copy3])
        temp[copy1++] = data[copy2++]; // copying the 1st half
    else
        temp[copy1++] = (data+n1)[copy3++]; // copying the 2nd half
}
while (copy2 < n1)
    temp[copy1++] = data[copy2++];
while (copy3 < n2)
    temp[copy1++] = (data+n1)[copy3++];
// copying the rest of remaining entries.

for(i=0; i<n1+n2; i++) // copying the data back
    data[i] = temp[i]; from temp.
delete [] temp; // releasing the temp's memory.
}
```

Problem 13.8

B

Problem 13.9

B

Problem 13.10

C

Problem 13.11

C

Problem 13.12

C

Problem 13.13

C

Problem 13.14

C

Problem 13.15

D

Problem 13.16

B

Problem 13.17

A

Problem 13.18

C

Problem 13.19

B