Assignment IV - DSAA(H)

Name: Yuxuan HOU (侯宇轩)

Student ID: 12413104

Date: 2025.10.11

Question 4.1 (0.1 marks)

Question 4.1 (0.1 marks) Say whether the following array is a Max-Heap (justify your answer):

34	20	21	16	14	11	3	14	17	13

Sol: Check relations:

- $34_1 > 20_2, 21_3$.
- $20_2 > 16_4, 14_5$.
- $21_3 > 11_6, 3_7$.
- $16_4 > 14_8$, but $16_4 \not> 17_9$, which violates the rule.
- $14_5 > 13_{10}$.

Due to the violation at 4 to 9, the array is not a Max-Heap.

Question 4.2 (0.1 marks)

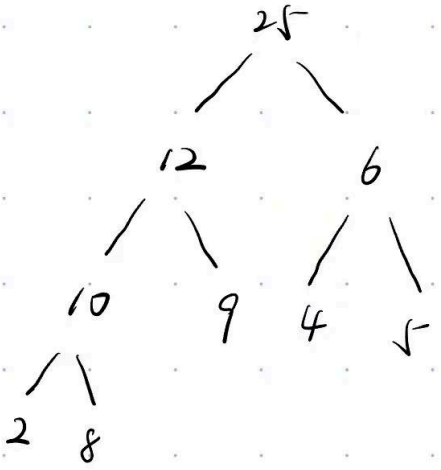
Consider the following input for HeapSort:

12	10	4	2	9	6	5	25	8

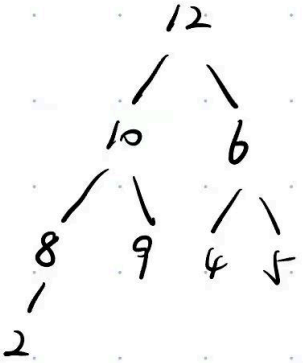
Create a heap from the given array and sort it by executing HeapSort. Draw the heap (the tree) after Build-Max-Heap and after every execution of Max-Heapify in line 5 of HeapSort. You don't need to draw elements extracted from the heap, but you can if you wish.

Sol:

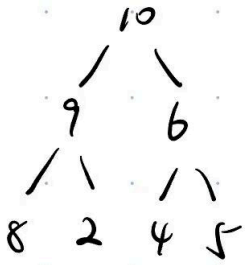
Max Heap:



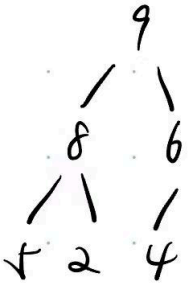
Heap Sort: I. 25



II. 12

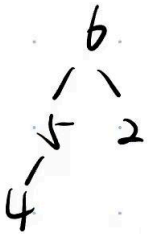


III. 10



IV. 9

V. 8



VI. 6

VII. 5

VIII. 4 2 2

Question 4.3 (0.5 marks)

1. Provide the pseudo-code of a MAX-HEAPIFY(A, i) algorithm that uses a WHILE loop instead of the recursion used by the algorithm shown at lecture.
2. Prove correctness of the algorithm by loop invariant.

Sol:

1.

Algorithm 1: MaxHeapify(A, i)

```
1  while true do
2  |  L ← 2 · i
3  |  R ← 2 · i + 1
4  |  largest ← i
5  |  if L ≤ A.heap_size and A[L] > A[largest] then
6  |  |  largest ← L
7  |  end
8  |  if R ≤ heap_size and A[R] > A[largest] then
9  |  |  largest ← R
10 |  end
11 |  if largest = i then
12 |  |  break
13 |  end
14 |  swap A[i] and A[largest]
15 |  i ← largest
16 end
```

2. **Loop invariant**: In each while iteration, the left and right subtrees of i are both Max-Heaps.

Initialization: At the beginning, the condition is trivally satisfy that the left and right subtrees of i are both Max-Heap.

Maintenance: If $largest = i$, then the subtree i will become a Max-Heap, otherwise, the possible violation i will be swaped to $largest$, and i was replaced by $largest$, and for the new i, it still follows the loop invariant.

Termination: When $largest = i$, the subtree of final i is a Max-Heap, thus the subtree of initial i will also become a Max-Heap. Plus, the termination is certain.

Question 4.4 (1.25 marks)

1. Show that each child of the root of an n-node heap is the root of a sub-tree of at most $(2/3)n$ nodes. (*HINT: consider that the maximum number of elements in a subtree happens when the left subtree has the last level full and the right tree has the last level empty. You might want to use the formula seen at lecture:* $\sum_{i=0}^{k-1} 2^i = 2^k - 1$).
2. As a consequence of (1) we can use the recurrence equation $T(n) \leq T(2n/3) + \Theta(1)$ to describe the runtime of Max-Heapify(A, n). Prove the runtime of Max-Heapify using the Master Theorem.

1. PF: Assuming the height is h. In the HINT condition, the size of left subtree is:

$$\sum_{i=0}^{h-1} 2^i = 2^h - 1$$

And for the right subtree, for its last level is empty, the size is:

$$\sum_{i=0}^{h-2} 2^i = 2^{h-1} - 1$$

The the total size is:

$$(2^h - 1) + (2^{h-1} - 1) + 1 = \frac{3}{2} \cdot 2^h - 1$$

Thus, we have:

$$\frac{2}{3}(\frac{3}{2} \cdot 2^h - 1) \geq 2^h - 1$$

Which is to say, the size of the subtree is at most $\frac{2}{3}n$ nodes.

`Q.E.D.`.

2. Sol: We have $a = 1, b = \frac{2}{3}, f(n) = \Theta(1)$.

Watershed: $n^{\log_b^a} = 1 = \Theta(1)$, thus let $k = 0$, $f(n) = \Theta(n^{\log_b^a} \lg^k n) = \Theta(1)$.

Therefore, $T(n) = \Theta(\log n)$.

Question 4.5 (1 mark)

Argue that the runtime of HeapSort on an already sorted array of distinct numbers is $\Omega(n \log n)$.

PF: For BuildMaxHeap, obviously its runtime aggregates to $\Theta(n)$.

To prove this, we can do the following calculation:

Let k be the height of a level, then there are approximately $\frac{n}{2^k}$ nodes at this height, then:

$$\sum_{k \geq 1} \frac{n}{2^k} \cdot O(k) = n \cdot O\Big(\sum_{k \geq 1} \frac{k}{2^k}\Big) = n \cdot O(2) = \Theta(n).$$

Then we will do some $\Theta(1)$ operations and Heapify $n-1$ times. At this time, for the array is sorted, the element swaped to the top will be heapify to the bottom, thus the cost of Heapify is $\Omega(height) = \Omega(\log n)$.

Then, we have:

$$\sum_{i=2}^{n} \Omega(\log i) = \Omega\Big(\sum_{i=2}^{n} \log i\Big) \geq \Omega\Big(\Big\lfloor \frac{n}{2} \Big\rfloor \cdot \log\Big(\frac{n}{2}\Big)\Big) = \Omega(n \log n).$$

Also, we can use the Stirling Formula: $\log(n!) = n \log n - n + O(\log n)$.

Eventually, the runtime is:

$$\Theta(n) + \Theta(1) \cdot (n-1) + \Omega(n \log n) = \Omega(n \log n)$$

`Q.E.D.`.

Question 4.6 (0.45 marks)

Implement HeapSort(A, n) and the two problems "Heap" and "Heap Operations" on the Judge system.

题目

状态	最后递交于	题目
✓ 100 Accepted	2 小时前	**26** Heap
✓ 100 Accepted	1 周前	**27** Heap Operations
✓ 100 Accepted	1 周前	**28** Heap Sort

```cpp
int main(){
    int T = read();
    while(T--){
        int N = read();
        vector < int > val(N + 10, 0), cur(N + 10, 0);
        for(int i = 1; i <= N; ++i)val[i] = read();
        for(int i = 1; i <= N; ++i)cur[i] = read();
        basic_string < char > ans;
        bool poss(true);
        for(int i = N; i >= 1; --i){
            int mx(INT_MIN), mn(INT_MAX);
            int mxHeap(-1), mnHeap(-1);
            for(int p = i; p >= 1; p >>= 1){
                if(cur[p] == val[i]){
                    if((p == 1 || cur[p >> 1] >= val[i]) && cur[p] > mx)mxHeap = p;
                    if((p == 1 || cur[p >> 1] <= val[i]) && cur[p] < mn)mnHeap = p;
                }mx = max(mx, cur[p]), mn = min(mn, cur[p]);
            }
            int res(-1);
            if(~mnHeap)ans += '0', res = mnHeap;
            else if(~mxHeap)ans += '1', res = mxHeap;
            else{poss = false; break;}

            int lst(cur[i]);
            for(int p = i; p > res; p >>= 1){
                int tmp = cur[p >> 1];
                cur[p >> 1] = lst;
                lst = tmp;
```

```
            }
        }
        if(!poss)printf("Impossible\n");
        else reverse(ans.begin(), ans.end()), printf("%s\n",
ans.c_str());
    }

    // fprintf(stderr, "Time: %.6lf\n", (double)clock() /
CLOCKS_PER_SEC);
    return 0;
}
```

```
int main(){
    int N = read();
    multiset < int > S;
    vector < string > res;
    while(N--){
        string opt; cin >> opt;
        if(opt == "insert"){
            int val = read();
            S.insert(val);
            res.push_back("insert " + to_string(val));
        }
        if(opt == "removeMin"){
            if(S.empty())res.push_back("insert 1");
            else S.erase(S.begin());
            res.push_back("removeMin");
        }
        if(opt == "getMin"){
            int val = read();
            while(!S.empty() && *S.begin() <
val)res.push_back("removeMin"), S.erase(S.begin());
            if(S.empty() || *S.begin() > val)res.push_back("insert " +
to_string(val)), S.insert(val);
            res.push_back("getMin " + to_string(val));
        }
    }
    printf("%d\n", (int)res.size());
    for(auto &s : res)cout << s << endl;
```

```

    // fprintf(stderr, "Time: %.6lf\n", (double)clock() / CLOCKS_PER_SEC);
    return 0;
}
```

```
int main(){
    int N = read();
    vector < int > A(N + 10, 0);

    for(int i = 1; i <= N; ++i)A[i] = read();
    #define LS (p << 1)
    #define RS (LS | 1)
    auto Heapify = [&](auto &&self, int p, int len)->void{
        int mx(p);
        if(LS <= len && A[LS] > A[mx])mx = LS;
        if(RS <= len && A[RS] > A[mx])mx = RS;
        if(mx != p)swap(A[mx], A[p]), self(self, mx, len);
    };
    auto HeapSort = [&](auto &&self, int len)->void{
        for(int i = (len >> 1); i >= 1; --i)Heapify(Heapify, i, len);
        for(int i = len; i > 1; --i)swap(A[1], A[i]), Heapify(Heapify, 1, i - 1);
    }; HeapSort(HeapSort, N);
    for(int i = 1; i <= N; ++i)printf("%d%c", A[i], i == N ? '\n' : ' ');

    // fprintf(stderr, "Time: %.6lf\n", (double)clock() / CLOCKS_PER_SEC);
    return 0;
}
```