hw4

資工三 110590002 王熯竑

1.

1.a.

minimum:O(1)
maximum:O(1)

1.b.

index of the corresponding element: $\text{index} + \lfloor \log(\text{index} + 1) \rfloor * 2(0.5 + \lfloor \log(\text{index} + 1) \rfloor - \lfloor \text{index} + \lfloor \log(\text{index} + 1) \rfloor + 1 \rfloor)$

1.c.

add new element to array last ,$O(1)$
make new element to be the right order in max heap or min heap, $O(\log(n))$
check the corresponding element in the other heap and swap if necessary, $O(1)$
if swap then repeat the process until the new element is in the right order. $O(\log(n))$

2.

2.a.

node of depth k in B_k is the handle, then T_0 is a single handle, is same as B_0 tree. T_1 is a handle node link with a parent, that is, handle node link a B_0 tree, is same as B_1 tree. T_2 is a handle link with a parent node, and handle's parent has a parenet with one another child that is, handle node link B_0 tree and link a B_1 tree, is same as B_2 tree. and so on.

2.b.

node of depth k in B_k is the handle, then T_0 is a single handle, T_1 is a handle node link with a parent, that is, handle node link a B_0 tree. that is, T_0 link a B_0 tree's root(r_0) T_2 is a handle link with a parent node, and handle's parent has a parenet with one another child that is, handle link r_0 and a B_1 tree(r_1). that is, T_1 link r_1. and so on.

3.

- step1: use quickselection find the element which has rank $\left\lfloor \frac{k-1}{2} \right\rfloor * \frac{n}{k}$ in O(n) time.
  that will split the array into two parts, S_1 is smaller than the element, and S_2 is larger than the element, and they have same size
- step2: repeat use step1 to get all the elements in S_1 and S_2 in O(n) time.: find from $\left(\frac{n}{k}\right)$ to $\left(\left\lfloor \frac{k-1}{2} \right\rfloor - 1\right) * \frac{n}{k}$ from S_1, find from $\left(\left\lfloor \frac{k-1}{2} \right\rfloor - 1\right) * \frac{n}{k}$ to $\lfloor k-1 \rfloor * \frac{n}{k}$ from S_2.
- time compelexity:
  $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$, for $n > k$ => $O(n \log(k))$
- example: array = [5,6,7,8,9,0,1,2,3,4] k=3 get target[3,6] by array[$i * \frac{n}{k} = 3$ for i in range(1,k)] get 6,[0,1,2,3,4,5],[7,8,9] by quickSelection target[floor(len(target)/2)] get 3,[0,1,2], [4,5] by quickSelection target[floor(floor(len(target)/2)/2)] no other index in target, return [3,6]

```
arr = input().split(',')
k= int(input())
```

```python
def sol(arr,kar):
    if len(arr)==0 or len(kar) ==0:
        return []
    k = kar[len(kar)//2]
    print(arr,kar)
    ( e,s1,s2 ) = quickSelection(arr,k)
    e1 = sol(s1,kar[:len(kar)//2])
    e2 = sol(s2,kar[len(kar)//2+1:])
    return [e]+ e1+e2
kar = []
for i in range(k):
    kar.append(int(len(arr)/k*(i+1)))
kar = kar[:-1]
print( sol(arr,kar))
```

4.

- step1: use quickSelection to find the median in O(n) time.
- step2: find all the elements and median distance in O(n) time.
- step3: use quickSelection to get the kth smallest distance in O(n) time.
- step4: find all element which distance which is smaller than kth smallest distance in O(n) time.
- time compelexity:
  $O(n)$

-example array=[9,5,8,7,6,4,3,2,1] k=3 get median 5, by quickselection get distance_array [4,0,3,2,1,1,2,3,4] get 3th_min_distance=1 get all element distance <= 3th_min_distance, [5,6,4]

```python
arr =[ int(i) for i in input().split(',')]
k= int(input())

median = quickSelection(arr,len(arr)//2)
distance = []
for i in arr:
    distance.append(abs(i-median))
kthDistance = quickSelection(distance,k)
ans= []
for i in range(len(arr)):
    if distance[i]<kthDistance and len(ans)!=k:
        ans.append(arr[i])
print(ans)
```