1. Proof by induction:

   Base case: when k=2, it is obvious that there will be a red node. The root is black, and the other one is red.

   Inductive hypothesis: Assume that for a red_black tree of k nodes, there exists at least a red node.

   Inductive step: Prove that for a red_black tree of k+1 nodes, there exists at least a red node.

   Case 1: Red node k+1 is inserted as a child of black node, this is the trivial case.

   Case 2: Red node k+1 is inserted as a child of red node, then there are three cases to solve red-red violation.

   Case1: the sibling of the parent of red node k+1 is red, then we will recolor them, but the red node k+1 will be red always, so we have at least one red node.

   Case2: the sibling of the parent of red node k+1 is black, and red node k+1 is an inside node of grandparent, then we will left rotate. Red node k+1 becomes an outside node of grandparent and remains red. Then go to case 3.

   Case3: the sibling of parent of red node k+1 is black, and red node k+1 is an outside node of grandparent, then we will right rotate the grandparent and recolor them, but the red node k+1 alse remains red after recoloring, so we also have at least a red node.

   Therefore, by induction, we can prove that red_black tree has at least one red node.

2. In the minimum case, we have one key in the root, then the other node have t-1 nodes, so the minimum number of keys is $1+(t-1)\sum_{i=1}^{h} 2t^{i-1}$,which t is 1000,h is 6, so the minimum number of keys is $2\times 10^{18}-1$.

   In the maximum case, we have 2t-1 keys in every node, so the maximum number of keys is $(2t-1)\ \sum_{i=0}^{h}(2t)^{i}$, so the maximum number of keys is $2^{7}\times 10^{21}-1$.

3. find minimum key:

   def search_min(T.root)

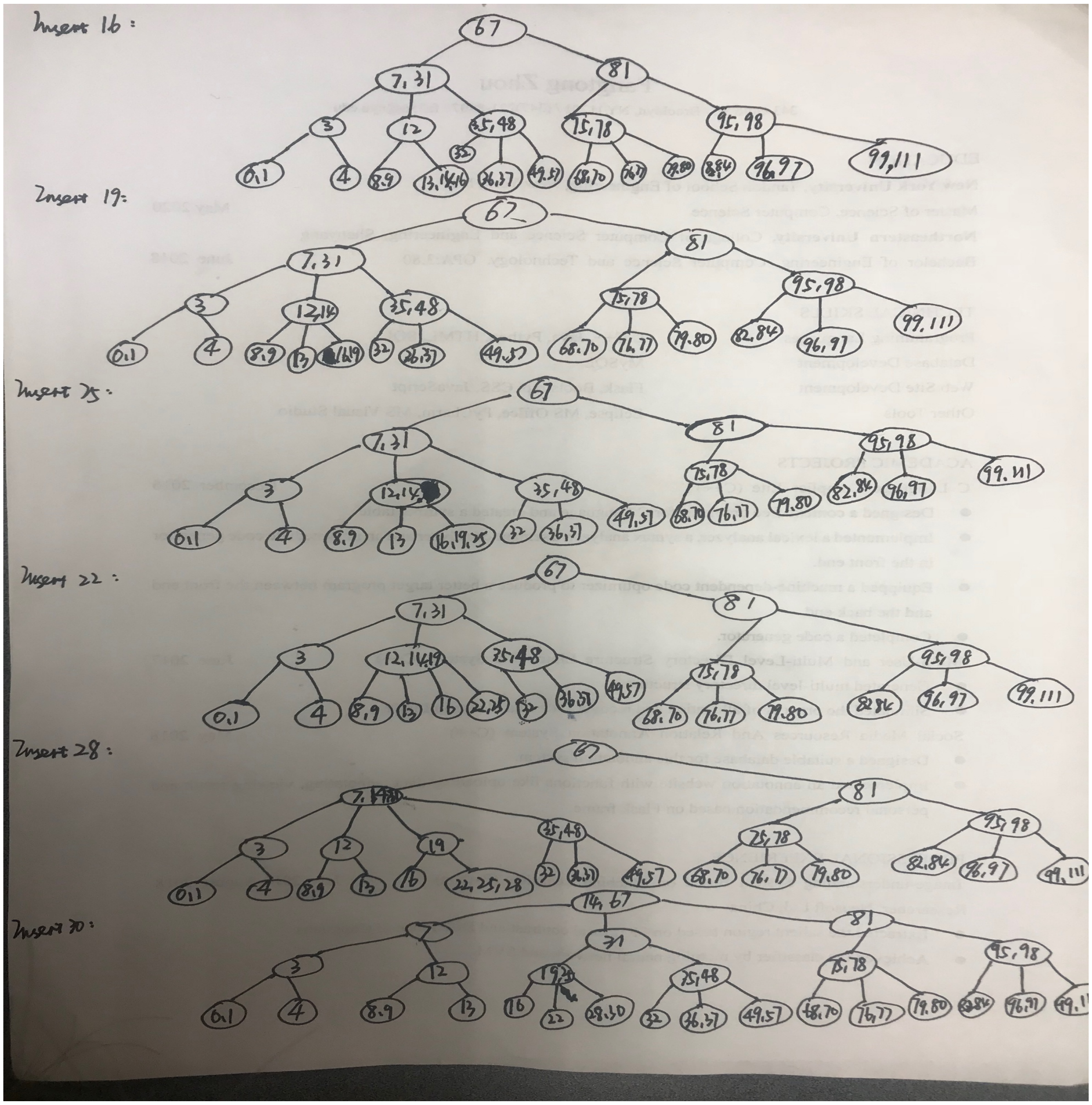
   x=T.root

   while($x.c_1$! $=$ NULL)

   x= $x.c_1$

```
return x.key_1
CPU running time : O(log_t n)
Disk access : O(log_t n)
find predecessor of a key:
def predecessor(T.root, k):
    (x.i)=B_Tree_Search(T.root,k)
    if(x.c_i == NULL)
        if(i!=1)
            return x.key_{i-1}
        else
            while x!=T .root
                x=x.p
                for i=1 to x.length
                    if x.key_i < k
                        return x.key_i
    else
        x=x.c_i
        while(x.c_{n+1} ≠ NULL)
            x= x.c_{n+1}
        return x.key_n
CPU running time : O(tlog_t n)
Disk access : O(log_t n)
```

4.



There are 6 disk access occurred when 16 was inserted.

5.Guess: $T(n) = O(nlog^2n)$

Check:

Inductive Hypothsis: T(k)= $O(klog^2k)$, $\forall k < n$

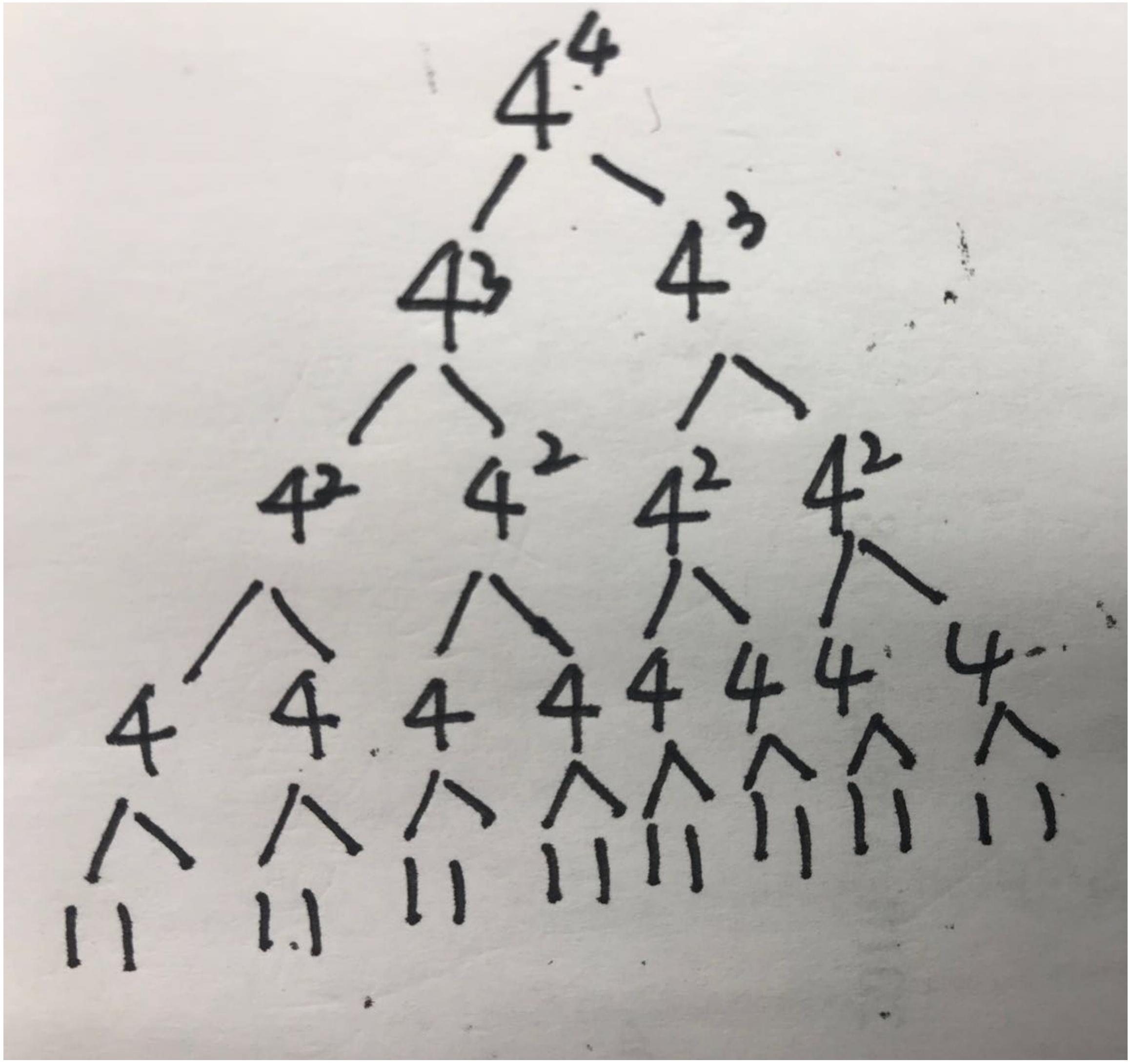
$$T(\text{n}) = 2T\left(\frac{n}{2}\right) + cnlogn$$

$$= 2\,O(\tfrac{n}{2}\log^2\tfrac{n}{2}) + cnlogn$$

$$= 2O(\tfrac{n}{2}(\log^2 n - 2logn + 1)) + cnlogn$$

$$= O(nlog^2n) + cnlogn = O(nlog^2n)$$

6.(1)

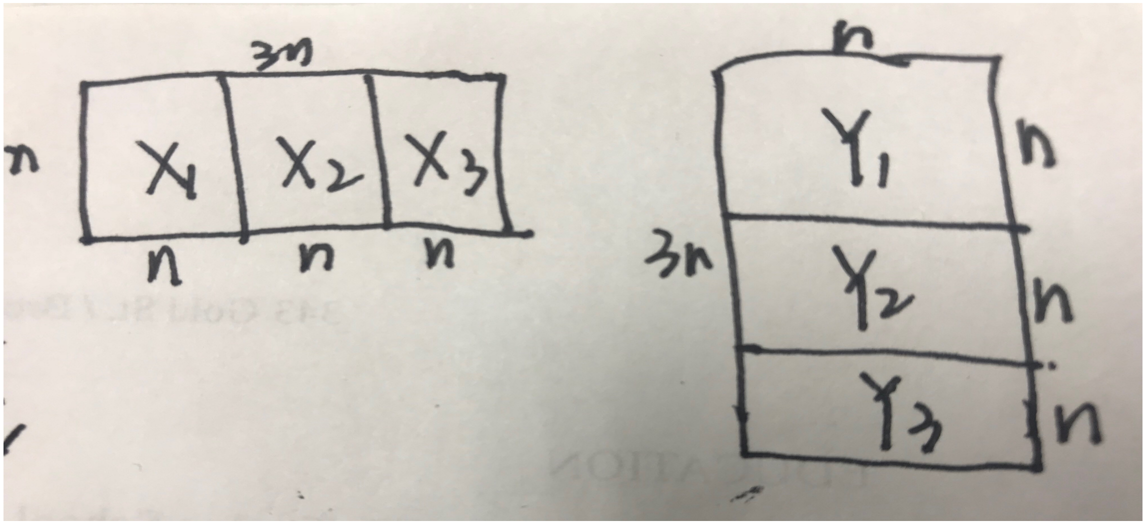


(2)$T(n) = 2T\left(\frac{n}{4}\right) + \Theta(n\mathrm{logn}) + \Theta(1)$

$$= \log_4 n\,\Theta(1) + nlogn\sum_{i=0}^{\log_4 n}\left(\frac{1}{2}\right)^i - n\log_4 n$$

$$= \Theta(nlogn)$$

7.



Just do like in the picture, multiply $X_1 and\, Y_1$, X_2 and Y_2 , X_3 and Y_3 by Strassen's algorithem. The Strassen's algorithm's running time is $O(n^{2.81})$, so this algorithm's running time $T(n) = 3O(n^{2.81}) = O(n^{2.81})$.

8. I designed a data structure which includes a Red-Black Tree and a variable Time which stores the smallest time difference between two ships.

already_reserved(t) is Red_Black tree search, so the running time is O(logn).

reserve(t) includes Red_Black Tree insert which running time is O(logn), and after inserting, need to find the new node's predecessor and successor in O(logn), then calculate their time difference, if either is smaller than Time, then update the variable Time, in that way, this method's running time is O(logn).

near_miss(), according to reserve method, we can get the smallest time difference in O(1).