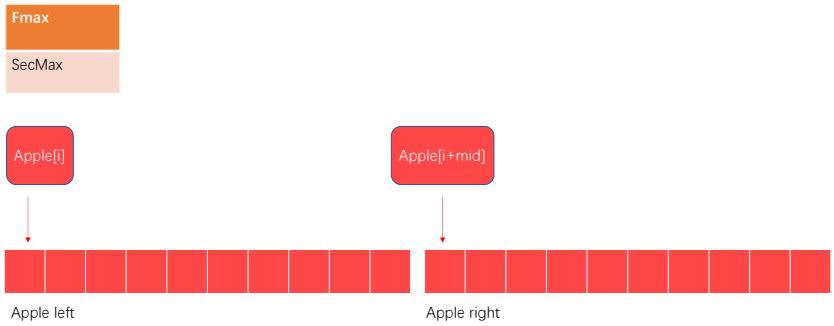
Yuan Gao z5239220 Q3

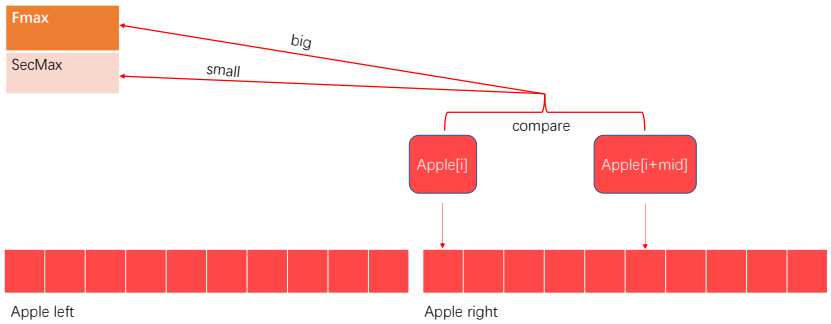
You are given 1024 apples, all of similar but different sizes and a small pan balance which can accommodate only one apple on each side. Your task is to find the heaviest and the second heaviest apple while making at most 1032 weightings. (20 points)

Step1: Divide the apples into two piles. Use **Fmax** to store the index of the heaviest apple and use secMax to store the index of the second heaviest apple.

Step2: Compare the weight of two apples. Heavy on the left and light on the right.



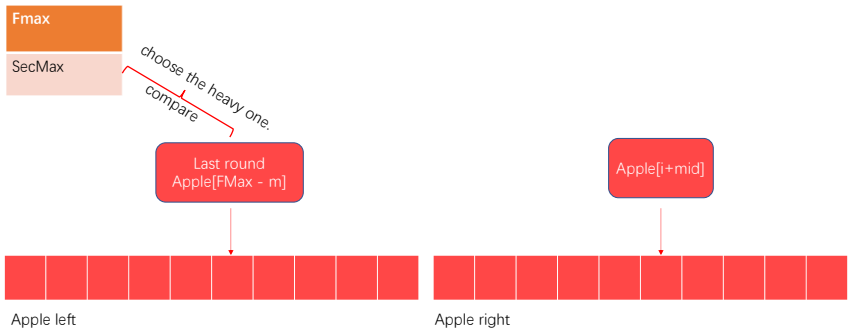
Step3: Recursion into the heavier side



The resulting Fmax is the heaviest apple but the second heaviest apple is not.

Therefore, we need compare to the second max.

Step4: Compare the small one and last round small apple, choose the heavy one.



Dichotomy and Dynamic programming

$$T(n) = T(\frac{n}{2}) + \frac{n}{2} + 1$$

$$= T(\frac{n}{4}) + \frac{n}{4} + \frac{n}{2} + 2$$

$$= T(\frac{n}{8}) + \frac{n}{8} + \frac{n}{4} + \frac{n}{2} + 3$$

$$= 2k - 1 + k - 1 = n + logn - 2$$

When $n = 1024$

$T(n) = 1032$

```cpp
//This is the cpp code
//*Swap two value**/
void swap(int * a, int * b){
    int tmp;
    *a = tmp;
    *a = *b;
    *b = tmp;
}

void Search(int *a, int &FMax, int &secMax, int left, int right){
    if(left == right){
        FMax = left;
        secMax = left;
       /*Recursive export*/
    }else if(left == right - 1){
        if(a[left] > a[right]){
            FMax = left;
            secMax = right;
        }else{
            FMax = right;
            secMax = left;
        }
    }else{
        /*Step1: Divide the apples into two piles.
           Use Fmax to store the index of the heaviest apple
           and use secMax to store the index of the second heaviest apple.*/
        int m = (right - left + 1) / 2;
        for(int i = left; i < left + m; i++){
```

```
29.             if(a[i] > a[i + m])
30.             /*Step2: Compare the weight of two apples.
31.                 Heavy on the left and light on the right.*/
32.                 swap(a[i], a[i + m]);
33.         }
34.         /*Step3: Recursion into the heavier side*/
35.         Search(a, FMax, secMax, left + m, right);
36.         /*Step4: Compare the small one and last round small apple, choose th
    e heavy one.*/
37.         if(a[FMax - m] > a[secMax])
38.             secMax = FMax - m;
39. }
```