CX2101 Algorithm Design and Analysis

Tutorial 1 (Sorting)

Week 4: Q4-Q6

This Tutorial

- Quicksort

- Algorithm Design

Question 4

- Suppose that, instead of using E[middle] as pivot, QuickSort also can use the median of E[first], E[(first + last)/2] and E[last]. How many key comparisons will QuickSort do in the worst case to sort *n* elements? (Remember to count the comparisons done in choosing the pivot.)
- **Ans**:
- Consider a subrange of *k* elements to be partitioned.
- Choosing the pivot requires 3 key comparisons in the worst case.
- Now $k - 3$ elements remain to be compared to the pivot.
  - It's easy to arrange not to compare elements that were candidates for the pivot again by swapping them to the extremes of the subrange
- So *k* comparisons are done by the time partition is finished.
- This is one more than if E[middle] is simply chosen as the pivot.

- An example of the worst case might be E[first] = 100, E[first + last/2] = 99, E[last] = 98, and all other elements are over 100.
- Then, 99 becomes the pivot, 98 is the only element less than the pivot, and k – 2 elements are greater than the pivot.
- The remaining keys can be chosen so that the worst case repeats – that is, we can arrange that the median-of-3 always comes out to be the second-smallest key in the range.
- Since the larger subrange is reduced by only 2 at each depth of recursion in QuickSort, it will require $n/2$ calls to partition(), with ranges of size n, $n-2$, $n-4$, …, 2 (suppose n is even), then the total number of comparisons is

$$f = n + (n-2) + (n-4) + \cdots + 2$$

$$f = 2 + 4 + \cdots + (n-2) + n$$

$$\therefore 2f = (n+2) + (n+2) + \cdots + (n+2) = \frac{n(n+2)}{2}$$

$n/2$ terms

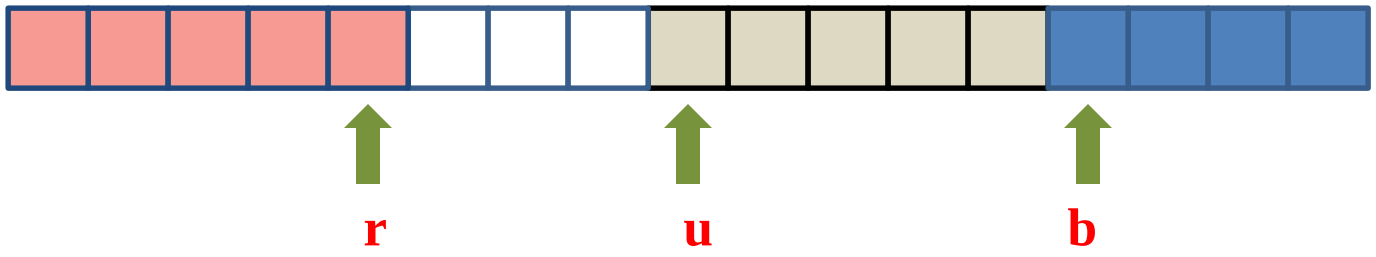
$$\therefore f = \frac{n^2 + 2n}{4} = \Theta(n^2)$$

- When n is odd, the total number of comparisons is also about $n^2/4$.
- So the worst-case time complexity with the median-of-3 strategy is $\Theta(n^2)$.

Question 5

- Each of n elements in an array may have one of the key values *red*, *white*, or *blue*.
- Give an efficient algorithm for rearranging the elements so that all the *reds* come before all the *whites*, and all the *whites* come before all the *blues*. (It may happen that there are no elements of one or two of the colours.)
- The only operations permitted on the elements are examination of a key to find out what colour it is, and a swap, or interchange, of two elements (specified by their indices).
- What is the asymptotic order of the worst case running time of your algorithm? (There is a linear-time solution.)

- We assume the elements are stored in the range 1, …, *n*
- At an intermediate step the elements are divided into four regions:
  - The first contains only reds
  - The second contains only whites
  - The third contains elements of colors unknown yet
  - The fourth contains only blues
- There are three indexes: r (last red), u (first unknown), b (first blue)



- The algorithm is:

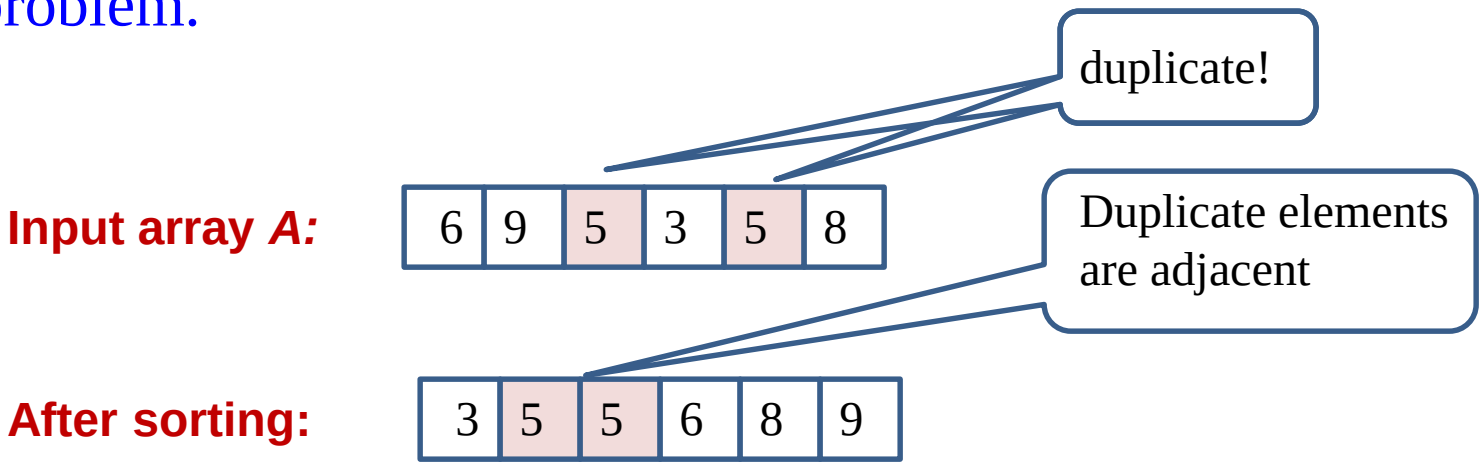
```
int r; // index of last red
int u; // index of first unknown
int b; // index of first blue

r = 0; u = 1; b = n+1;
while (u < b)
        if (E[u] == red)
                Interchange E[r+1] and E[u];
                r ++;
                u ++;
        else if (E[u] == white)
                u ++;
        else // (E[u] == blue)
                Interchange E[b-1] and E[u]
                b --;
```

- With each iteration of the while loop, either u is increased by one or b is decreased by one, so there are n iterations.
- Each iteration takes constant time, so the overall time is linear $\Theta(n)$.

Question 6

- Suppose we have an unsorted array *A* of *n* elements and we want to know if the array contains any duplicate elements.

(a) Outline (clearly) an efficient method for solving this problem.



- **Ans**: First, sort the array in increasing (or decreasing) order. Then, do a post-scan of the sorted array to check for equal adjacent elements.

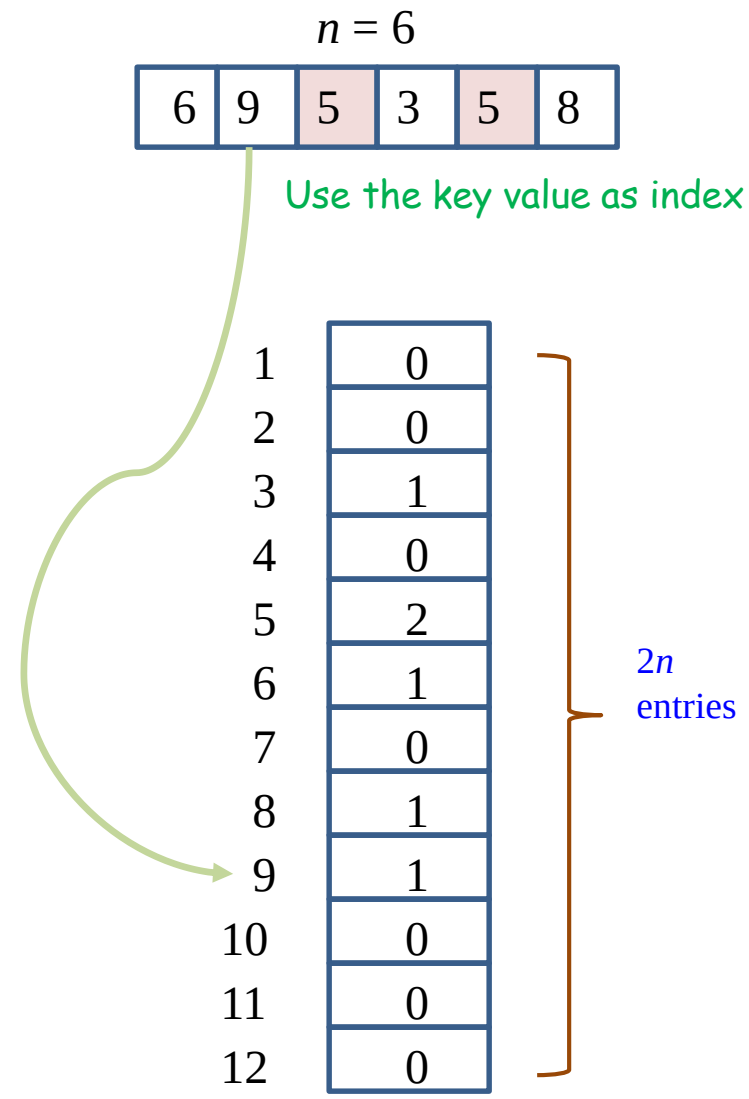
Question 6

```
Bool ExistDuplicate(int A[], int n) {
     MergeSort(A, n); // or Heapsort
     for (int i = 1; i < n; i++) {
            if (A[i] == A[i-1])
                  return true;
     }
     return false;
}
```

- (b) What is the asymptotic order of the running time of your method in the worst case? Justify your answer.
- **Ans**: The worst-case time for sorting is $\Theta(n\lg n)$ using Mergesort (or Heapsort). The time for post-scan is $\Theta(n)$. Therefore, the overall worst-case running time of the algorithm is $\Theta(n\lg n)$.

Question 6

(c) Suppose we know the n elements are integers from the range 1, …, $2n$, so other operations besides comparing keys may be done. Give an algorithm for the same problem that is specialized to use this information. Tell the asymptotic order of the worst case running time for this solution. It should be of lower order than your solution for part (a).



- **Ans**: Simply count how many have each key value, using an array of $2n$ entries to store the counts.
- We can quit as soon as any count becomes 2.
- Since we scan the n-elements array once, and each element takes constant time to count, the overall worst-case running time is $\Theta(n)$.

What we have exercised

- Quicksort
  - Pivot selection and its effect to time complexity
- Algorithm Design
  - Use/Extend some functions in sorting algorithms to solve new problems
  - Use sorting algorithms for problem solving
  - Trade space for time