CX2101 Algorithm Design and Analysis

Tutorial 1 (Sorting)

Week 3: Q1-Q3

This Tutorial

- Insertion Sort
- Divide and Conquer Approach
- Mergesort

Question 1

- The worst case for Insertion Sort occurs when the keys are initially in decreasing order. Suppose the performance of Insertion Sort is measured by the total number of comparisons between array elements (not counting the number of swaps). Show at least two other initial arrangements of keys that are also worst cases for Insertion Sort.
- **Ans**: Two other worst-case input for Insertion sort are:

$$(n, n-1, n-2, \ldots, 3, 1, 2)$$
$$(1, n, n-1, n-2, \ldots, 2)$$

Note: In each of above lists, the ***i*th key still requires *i* comparisons** to find its correct position, although a comparison may not be followed by a **swap** of keys.

Question 2

- Use the divide and conquer approach to design an algorithm that finds both the largest and the smallest elements in an array of n integers. Show that your algorithm does at most roughly $1.5n$ comparisons of the elements. Assume $n = 2^k$.
- Ans:

```
void findMinMax(int[] ar, int start, int end, int[] minMax)
// minMax is an array of two integers to return the minimum
// and maximum found
{
     int tempMinMax1[] = new int[2];
     int tempMinMax2[] = new int[2];
     int mid;

     if (start == end) { // one element array
             minMax[0] = ar[start];
             minMax[1] = ar[start];
     }
```

```
    else if (end - start == 1) { // two element array
            if (ar[start] > ar[end]) {
                    minMax[1] = ar[start];
                    minMax[0] = ar[end];
            } else {
                    minMax[0] = ar[start];
                    minMax[1] = ar[end];
            }
    }
    else {  // array longer than two elements
            mid = (start + end)/2;

            findMinMax(ar, start, mid, tempMinMax1);
            findMinMax(ar, mid+1, end, tempMinMax2);

            if (tempMinMax1[0] < tempMinMax2[0])
                    minMax[0] = tempMinMax1[0];
            else    minMax[0] = tempMinMax2[0];

            if (tempMinMax1[1] > tempMinMax2[1])
                    minMax[1] = tempMinMax1[1];
            else    minMax[1] = tempMinMax2[1];
    }
}
```

Question 2

- To show that the above algorithm does at most roughly $1.5n$ comparisons of the elements (assume $n = 2^k$):
- Let W_i be the number of comparisons for i elements, then the recurrence equation is:

$$W_1 = 0$$
$$W_2 = 1$$
$$W_n = W_{n/2} + W_{n-n/2} + 2$$

- If $n = 2^k$, where k is a positive integer,

$$W_n = 2W_{n/2} + 2$$

$$W_n = 2W_{n/2} + 2$$
$$= 2\left(2W_{n/2^2} + 2\right) + 2$$
$$= 2^2 W_{n/2^2} + 2^2 + 2$$
$$= 2^2\left(2W_{n/2^3} + 2\right) + 2^2 + 2$$
$$= 2^3 W_{n/2^3} + 2^3 + 2^2 + 2$$
$$\dots$$
$$= 2^{k-1} W_{n/2^{k-1}} + 2^{k-1} + 2^{k-2} + \cdots + 2^2 + 2$$
$$= 2^{k-1} + 2^{k-1} + 2^{k-2} + \cdots + 2^2 + 2$$
$$= 2^{k-1} + 2(\underbrace{2^{k-2} + 2^{k-3} + \cdots + 2 \ + 1}_{\text{Geometric series}})$$
$$= 2^{k-1} + 2(\underline{2^{k-1} - 1})$$
$$= 2^{k-1} + 2^k - 2$$
$$= n/2 + n - 2 = \frac{3}{2} n - 2$$

Since $n = 2^k$
So $n/2^{k-1} = 2$
$W_2 = 1$

$$\frac{1(2^{k-1} - 1)}{2 - 1}$$

Question 3

- Show how MergeSort sorts each of the arrays below and give the number of comparisons among array elements in sorting each array.

(1) 14 40 31 28 3 15 17 51

14 28 31 40 3 15 17 51

Ans: The array is first divided into two equal parts

14	40	31	28

3	15	17	51

Each part is then sorted by MergeSort. The process begins by dividing each part into equal parts

14	40

31	28

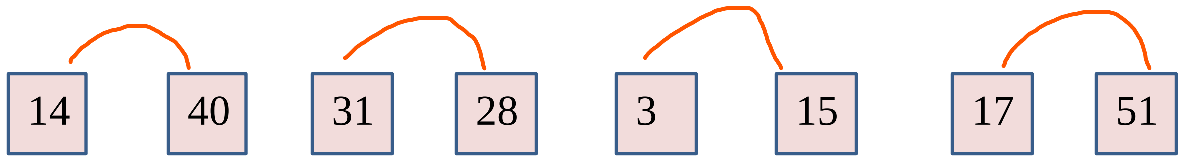
3	15

17	51

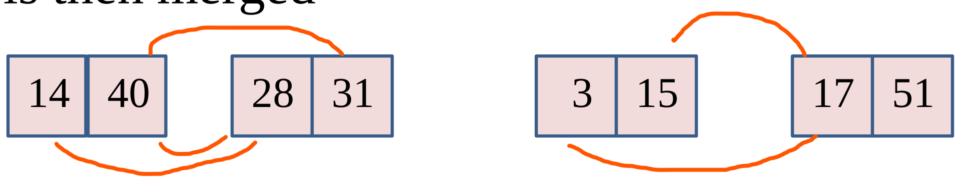
and then each of these parts into equal parts

14	40	31	28	3	15	17	51

This subdivision process now ends because each part contains only one item.

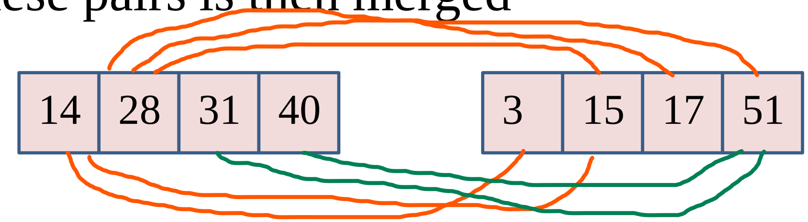


Each pair is then merged



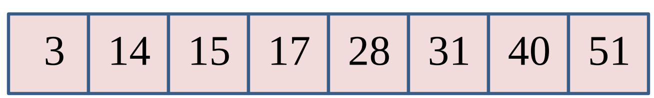
4 comparisons

Each of these pairs is then merged



5 comparisons

Finally these pairs are merged



7 comparisons

to obtain the sorted array. Totally, it takes 4 + 5 + 7 = 16 comparisons.

Question 3

- Show how MergeSort sorts each of the arrays below and give the number of comparisons among array elements in sorting each array.

(2) 23 23 23 23 23 23 23 23 + 4

(2) **Ans**: The array is first divided into two equal parts

23	23	23	23

23	23	23	23

Each part is then sorted by mergesort. The process begins by dividing each part into equal parts

23	23

23	23

23	23

23	23

and then each of these parts into equal parts

23 23 23 23 23 23 23 23

This subdivision process now ends because each part contains only one item.

23	23	23	23	23	23	23	23

Each pair is then merged

23	23	23	23	23	23	23	23

4 comparisons

Each of these pairs is then merged

23	23	23	23	23	23	23	23

4 comparisons

Finally these pairs are merged

23	23	23	23	23	23	23	23

4 comparisons

to obtain the sorted array. Totally, it takes 4 + 4 + 4 = 12 comparisons.

What we have exercised

- Insertion Sort
  - Worst case: descending order and its variants
- Divide and Conquer Approach
  - Terminating condition
  - Divide and combine
  - Complexity analysis by solving recurrence equation
- Mergesort
  - Divide and conquer
  - Complexity analysis by counting the number of comparisons