Jesus Ugarte

University of Central Florida

Computer Science I

Sorting Exercise

Show the contents of the array below being sorted using Insertion Sort at the end of each loop iteration.

Initial	2	8	3	6	5	1	4	7
	2	8	3	6	5	1	4	7
	2	3	8	6	5	1	4	7
	2	3	6	8	5	1	4	7
	2	3	5	6	8	1	4	7
	1	2	3	5	6	8	4	7
	1	2	3	4	5	6	8	7
Sorted	1	2	3	4	5	6	7	8

2) Show the contents of the array below being sorted using Selection Sort at the end of each loop iteration. As shown in class, please run the algorithm by placing the smallest item in place first.

Initial	6	2	8	1	3	7	5	4
	1	2	8	6	3	7	5	4
	1	2	8	6	3	7	5	4
	1	2	3	6	8	7	5	4
	1	2	3	4	8	7	5	6
	1	2	3	4	5	7	8	6
	1	2	3	4	5	6	8	7
Sorted	1	2	3	4	5	6	7	8

3) Show the contents of the array below being sorted using Bubble Sort at the end of each loop iteration. As shown in class, please run the algorithm by placing the largest item in place first.

Initial	4	2	6	5	7	1	8	3
	2	4	5	6	1	7	3	8
	2	4	5	1	6	3	7	8
	2	4	1	5	3	6	7	8
	2	1	4	3	5	6	7	8
	1	2	3	4	5	6	7	8
	1	2	3	4	5	6	7	8
Sorted	1	2	3	4	5	6	7	8

4) When Merge Sort is run on an array of size 8, the merge function gets called 7 times. Consider running Merge Sort on the array below. What would the contents of the array be right before the 7^{th} call to the Merge function?

Initial	7	2	1	5	8	3	4	6
Before 7^{th} Merge	1	2	5	7	3	4	6	8

5) Show the result of running Partition (as shown in class on Friday) on the array below using the leftmost element as the pivot element. Show what the array looks like after each swap.

PIVOT

Initial	5	2	1	7	8	3	4	6
	5	2	1	4	8	3	7	6
	5	2	1	4	3	8	7	6
After Partition	3	2	1	4	5	8	7	6

6) Show the contents of the array below after each merge occurs in the process of Merge-Sorting the array below:

Initial	3	6	8	1	7	4	5	2
1	3	6	8	1	7	4	5	2
2	3	6	8	1	7	4	5	2
3	3	6	8	1	7	4	5	2
4	3	6	8	1	7	4	5	2
5	3	6	1	8	4	7	2	5
6	1	3	6	8	2	4	5	7
Last	1	2	3	4	5	6	7	8

7) Here is the code for the partition function (used by Quick Sort). Explain the purpose of each line of code.

```
int partition(int* vals, int low, int high) { // main function to be Called

int lowpos = low; // initialize Value of Current lowest Position

low++; // increment Value of low by one

while (low <= high) // Set Condition, repeat loop while low less or equal to high

{

    while (low <= high && vals[low] <= vals[lowpos]) // while low Value less than or equals to high and Value of Current low in array is less than or equal to Curent lowesPosition

          low++; // increment low by 1

        while (high >= low && vals[high] > vals[lowpos])
                                                        // if high is greater than or equal to low and Current Value of high is greater than Current Value in lowest Position.
              high--; // decrement high by 1

            if (low < high)
                 swap(&vals[low],&vals[high]);
                                                        // Swap if lowest is less than high
  }
swap(&vals[lowpos], &vals[high]);
return high; // Swap Current Low position and Current Val.


}
```

8) Explain, why in worst case scenario the quick sort algorithm runs more slowly than Merge Sort

The answer depends on the Pivot we choose. Quick sort woks best with large data sets. Merge sort runs faster because it does not Partition but starts splitting.

9) In practice, quick sort runs slightly faster than Merge Sort. This is because the partition function can be run "in place" while the merge function can not. More clearly explain what it means to run the partition function "in place".

Partition ensures that all items less than the Pivot Precede it and returns the Position of the Pivot. this meets our Condition for the Problem.

10. You are trying to write a code for selection sort and you come-up with the following code. However, there is a bug in the code. Identify that bug and explain why that is a bug and edit that part of the code to correct it. Later, analyze the run-time of the updated code:

```
void selectionSort(int arr[], int n)
{
    int i, j, min_idx, temp;

    // One by one move boundary of unsorted subarray
for (i = 0; i < n-1; i++)
    {
        min_idx = i;
        for (j = 0;) j < n; j++)
        if (arr[j] < arr[min_idx])
        min_idx = j;

        temp = arr[i];
        arr[i] = arr[min_idx];
        arr[min_idx] = temp;

    }
}
```

bug

the bug is present because it is neccesary to analize one element after the initial index hence the correct initialization of j variable is (J = i+1)

Run Time: Since it is a nested loop

$(n * n) \rightarrow O(n^2)$

for selection sort

11) Explain the steps to come-up with the recurrence relation for merge sort and solve the recurrence relation to get the run-time of merge sort.

recurrence relation for merge Sort

$T(n) = T(n/2) + T(n/2) + O(n)$

$T(n) = 2T(n/2) + n$

best Case

$T(1) = 1$

$T(n) = 4T(n/4) + n + n$

$T(n) = 8T(n/8) + n + n + n$

$T(n) = 8T(n/8) + 3n$

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

$= 2\frac{n}{4} + n/2$

$T(n/4) = 2(n/8) + n/4$

$T(n/8) = 2(n/16) + n/8$

PATTern!

$T(n) = 2^k T(n/2^k) + kn$

$T(n) = 2^{\log n} T(2^k/2^k) + (\log n)n$

$T(n) = n\,T(1) + (\log n)n = n + n\log n$

$T(n) = n + \log n$