Rajshahi University of Engineering and Technology.

Course No: CSE-2201

Course Title: Sessional Based on CSE-2201

Lab report-3

Submitted to,

Dr. Md-Ali Hossain,
Associate professor,
Department of computer Science & Engineering, RUET.

Submitted by,

Md-AL-Amin Tokder Shoukhin

Roll: 1803078

Sec: B
Department of computer Science & Eng., RUET.

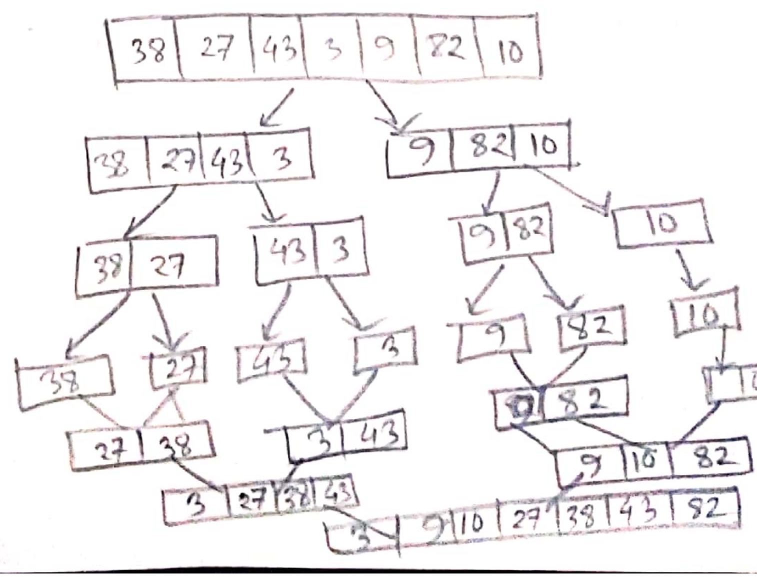
Scanned with CamScanner

Title: Find out the time & space complexity of the Merge sort approach and compagre it with the performance of Quick sort algorithm for (i) Best (ii) Average (iii) Worst case.

Introduction: Merge sort & quie sort are external algorithm which are based on devide & conquer strategy. The Quick sort complexity in best case is $\Omega(n\log(n))$, average case complexity is $\Theta(n\log(n))$ and Worst case complexity is $O(n^2)$. On, the other hand Merge sort best cas complexity is $\Omega(n\log n)$ average case is $\Theta(n\log n)$ and worst case is $O(n\log n)$.

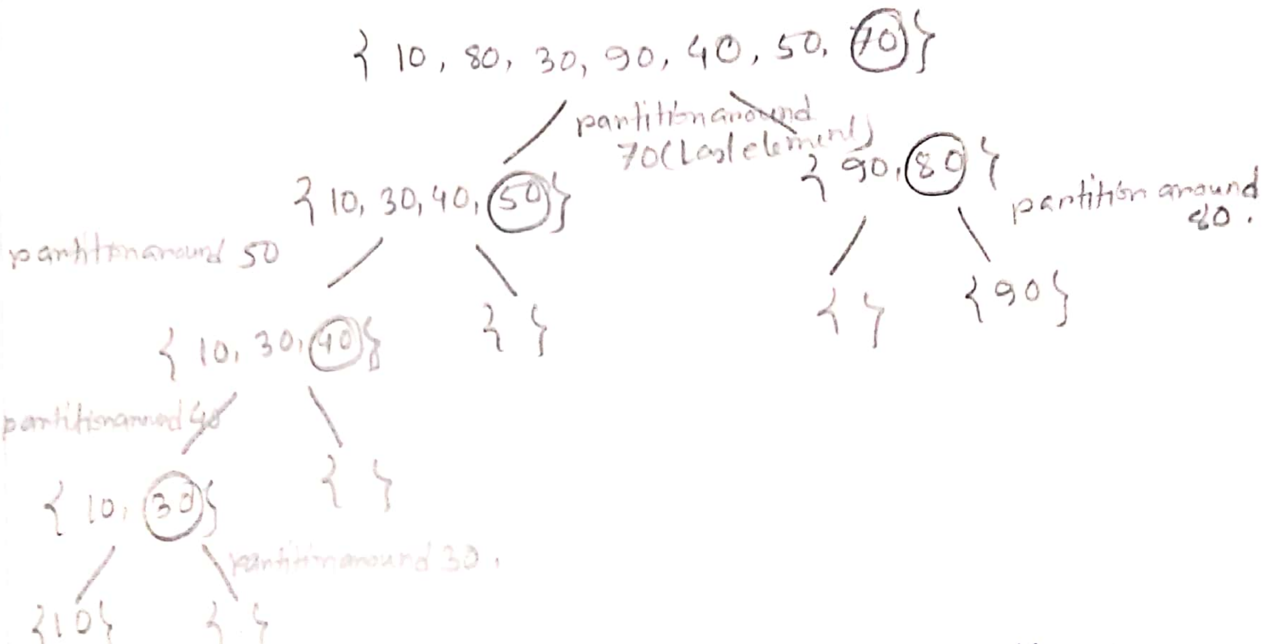
Description: Merge sort is a devide and conquer algorith. It divides the Input array into two halves, calls itself for the two halves and then merges the two sorted halves. The merge() function is used for merging two halves.

for example,



On the other hand, Quick sort is also an divide & conquer Algorithm. It picks an element as pivot & partitions the given array arround the picked pivot.

For example,



Time complexity & Space complexity of Each Algorithm:

Here is the recurrence relation of merge sort,

$$T(n) = \begin{cases} a & , n = 1, \text{ a a constant} \\ 2T(n/2) + cn & n > 1, \text{ c a constant} \end{cases}$$

when n is a power of 2, $n = 2^k$. then,

$$T(n) = 2(2T(n/4) + cn/2) + cn$$
$$= 4T(n/4) + 2cn.$$
$$= 4(2T(n/8) + cn/4) + 2cn$$
$$\vdots$$
$$= 2^k T(1) + kcn$$
$$= an + cn\log n.$$

$\therefore$ $T(n)$ is $O(n\log n)$.

For space complexity,

Mergesort recursive function will work like this,

$$n \rightarrow n/2 \rightarrow n/4 \rightarrow n/8 \rightarrow \cdots\cdots 1$$

From this perspective the merge sort is taking K logn space.
But there is an additional array or space required to copy the sorted elements of the subarrays and Kn space is required to copy the two subarrays having n/2 elements to get the sorted n elements.

$\because$ Kn > Klogn

So the mergesort space complexity is $O(n)$.

In Quicksort algorithm,

Here the recurrence relation that maintain in Quicksort,

$$T(n) = n+1) + \frac{1}{n} \sum_{1 \le k \le n} [T(k-1) + T(n+k)]$$

In worst case,
$$T(1) = c_1$$
$$T(n) = T(n-1) + cn$$
$$= T(n-2) + 2cn - c$$
$$= T(n-3) + 3cn - 3c$$
$$\vdots$$
$$= T(n-k) + kcn - \frac{k(k-1)}{2} c.$$

If $n = k$, $T(n) = T(1) + n^2c - \frac{n^2+n}{2} \cdot c$

$\therefore$ Quick sort has complexity $O(n^2)$ in worst case

But in average case,

$$T(1) = C_1$$

$$T(n) = \frac{1}{n} \sum_{k=0}^{n-1} \{T(k-1) + T(n-k)\} + cn.$$

By solving the recurrence relation we get $O(n \log n)$

For space complexity in quick sort,

$$S(n) \leq \begin{cases} 2 + S(\lfloor (n-1)/2 \rfloor), & n > 1 \\ 0 & n \leq 1 \end{cases}$$

So the space complexity is $O(\log n)$.

But in worst case scenario it can be $O(n)$ and in this case this sort avoided ~~to~~ randomized partition.

Algorithm:

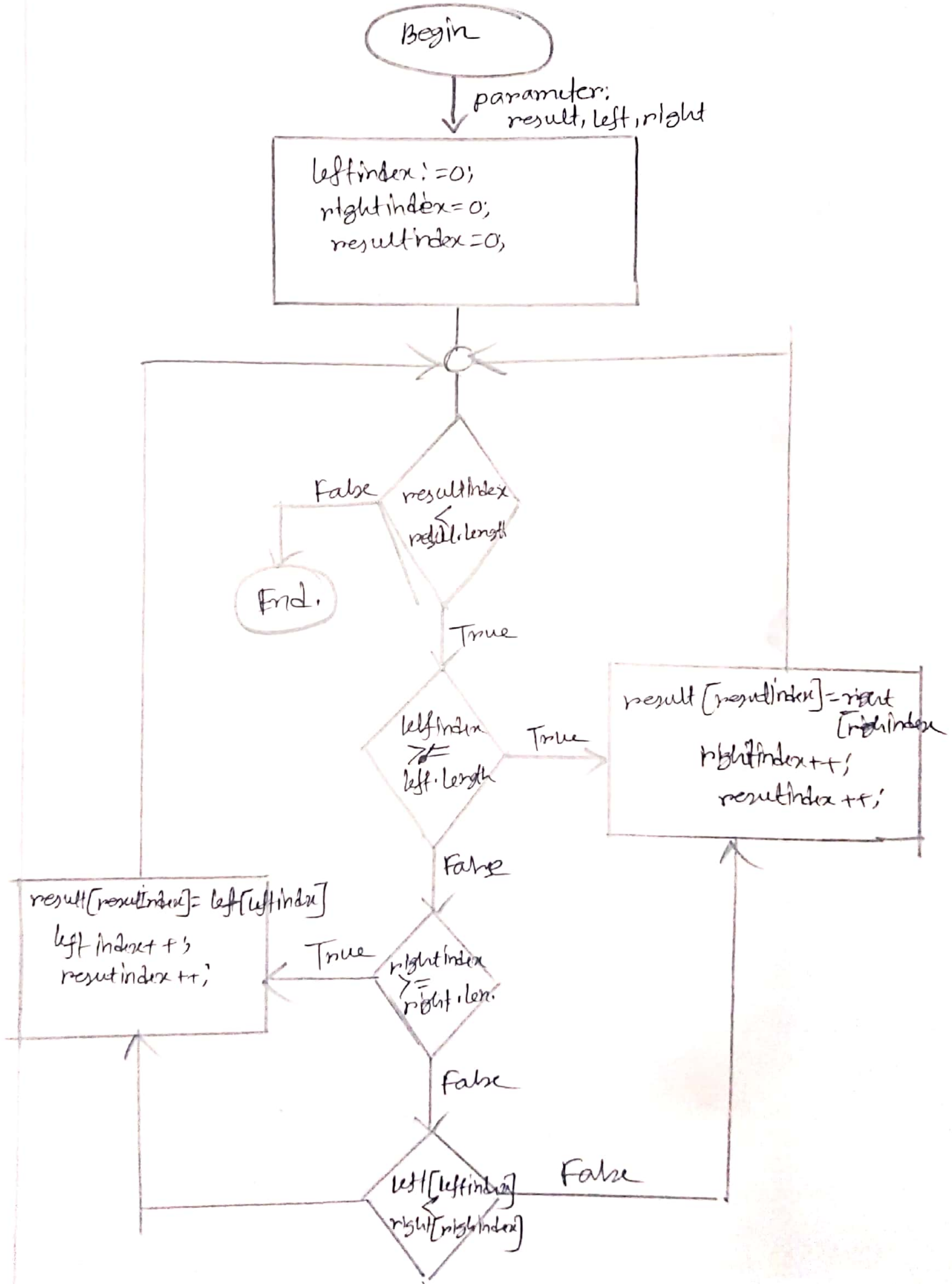
```
Algorithm MergeSort (low, high)
{ if (low < high) then
    { mid = ⌊(low + high)/2⌋;
      MergeSort (low, mid);
      MergeSort (mid+1, high);
      Merge (low, mid, high);
    }
}
```

A Algorithm Merge (low, mid, high)

```
{ h := low, i := low; j := mid + 1;
  while ((h ≤ mid) and j ≤ high) do
  { if (a[h] ≤ a[j]) then
      { b[i] := a[h]; h := h + 1;
      }
    else
      { b[i] := a[j]; j := j + 1;
      }
      i := i + 1;
  }
  if (h > mid) then
      for k := j to high do
          { b[i] := a[k]; i := i + 1;
          }
  else
      for k := h to mid do
          { b[i] := a[k]; i := i + 1;
          }
      for k := low to high do a[k] := b[k];
}
```

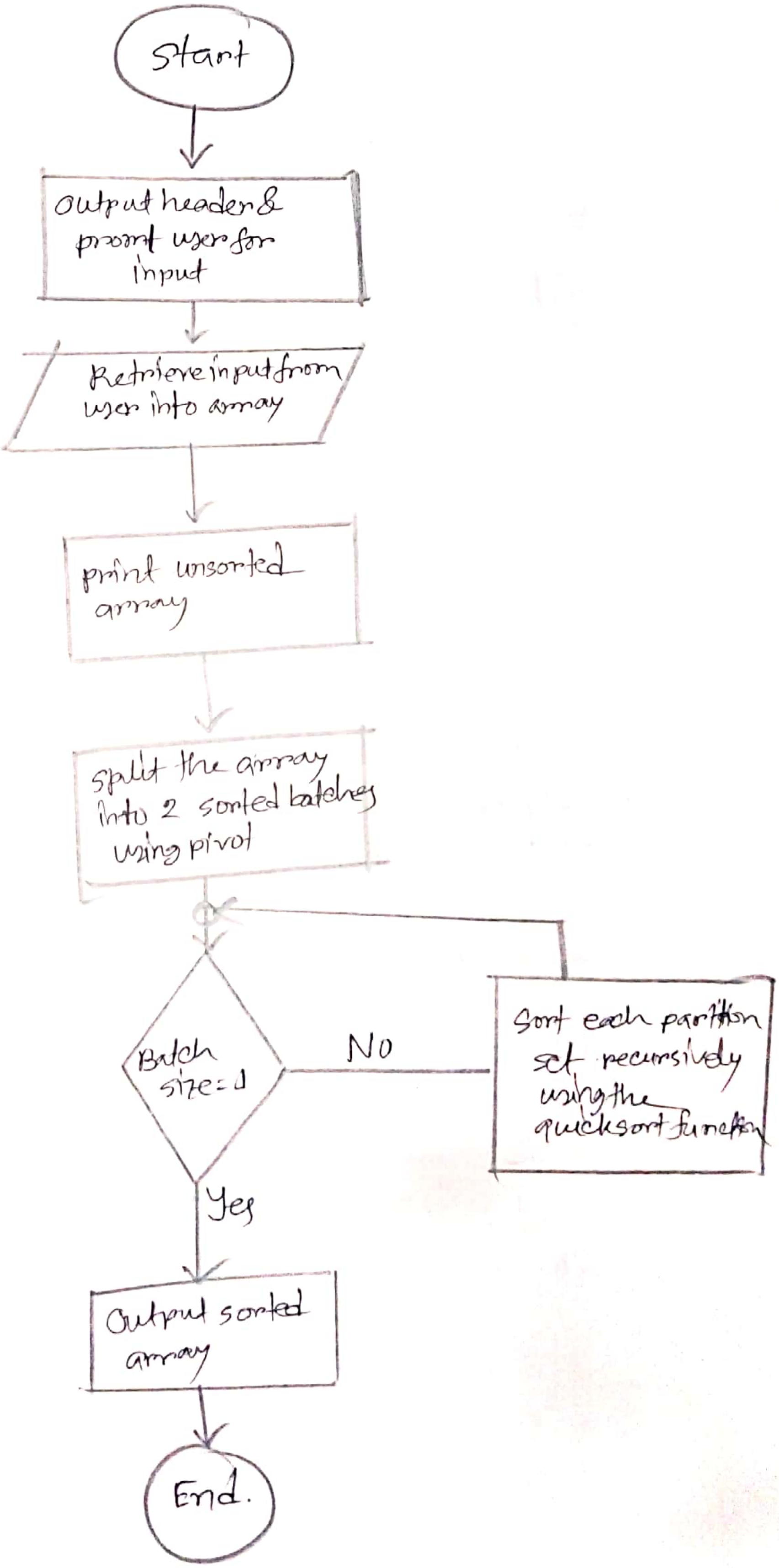
```
Algorithm QuickSort (p, q)
{ if (p<q) then
   {  j := Partition (a, p, q+1);
      QuickSort ( p, j-1);
       QuickSort (j+1, q);

   }
}

Algorithm Partition (a, m, p)
{ ~~for~~(
   v:= a[m] ; i := m; j := p;
   repeat
    { repeat
           i := i+1;
         until (a[i] >= v);

         repeat
            j := j-1;
            until (a[j] <= v);
         if (i<j) then Interchange (a, i, j);

    } until (i >= j);
    a[m] := a[j]; a[j] := v; return j;

}
Algorithm interchange (a, i, j) { p := a[i];
                                   a[i] := a[j]; a[j] := p; }
```

Flowchart(For Mergesort):



Scanned with CamScanner

Flowchart (for QuickSort):



~~Conclusion~~:

Input and output:

Input:

4000

2198 3763 2469 .... (upto 4000 elements)

Output:

Sorted array by Merge sort is:

2 4 5 8 10 12 12 12 16 .... (upto 4000)

sorted array by Quick sort is: (upto 4000)

2 4 5 8 10 12 12 12 16 ......

Number of exchange operation in Merge sort is: 120257

Number of exchange operation in Quicksort is: 92496.

Number of exchange operation in

Conclusion:

By compare the exchange operation from ~~above~~ graph we can say that Merge sort is more efficient for large data size and its has O(n logn) complexity for best, worst and avg case also. Quick sort performs well for short size data and it also perform in n logn time thats why. merge short is more efficient (for large data) than ~~that~~ Quicksort.