DON’T WASTE YOUR TIME

- **Course:** CS 210
- **Section:** 815 and 1375
- **By:**

Norah AlRubayan (220410543)
Shoug AlHargan (220410521)
Shajan Mokyel (221410469)

Sorting
time

Introduction to Sorting

What does sorting mean?

It means to arrange something in a meaningful order.

Example:

Recycling is a type of sorting we sort trash depending on its category.

Software and sorting:

Even in technology we use different sorting ways to arrange our information (Data).

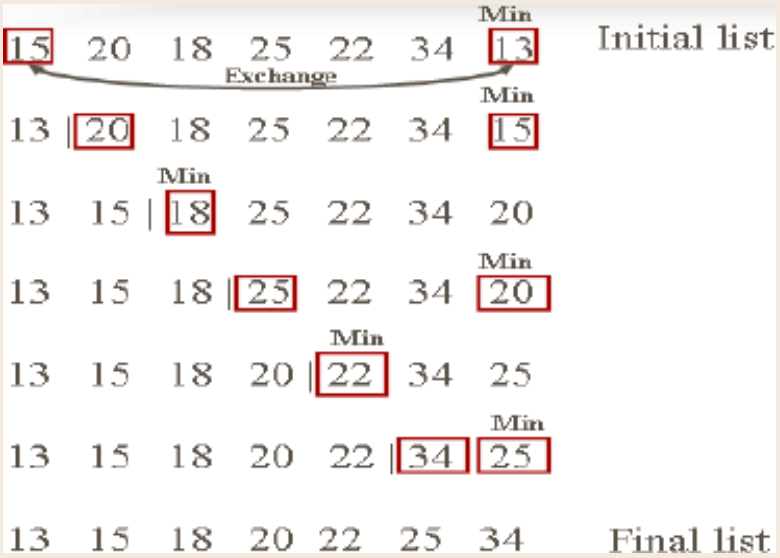
Some Types of Sorting in software:

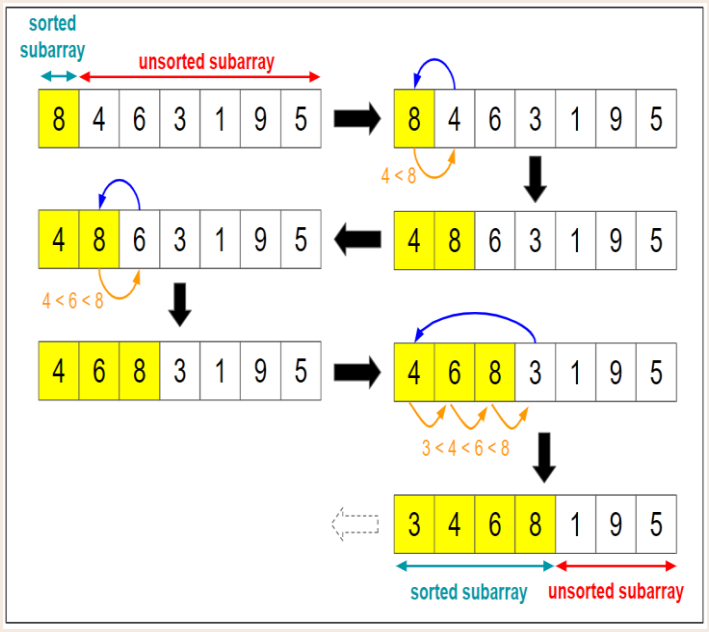
- **Merge Sort**
- **Insertion Sort**
- **Bubble Sort**
- **Selection Sort**
- **Quick Sort**

Sorting Algorithms

Selection Sort

This sorting method works by selecting the first value of a group of data as being the smallest value. Then it compares this value with the rest of the data and if a smaller value was found then this small value will be placed as the first value of the data. This is repeated until all the data is organized in ascending order.



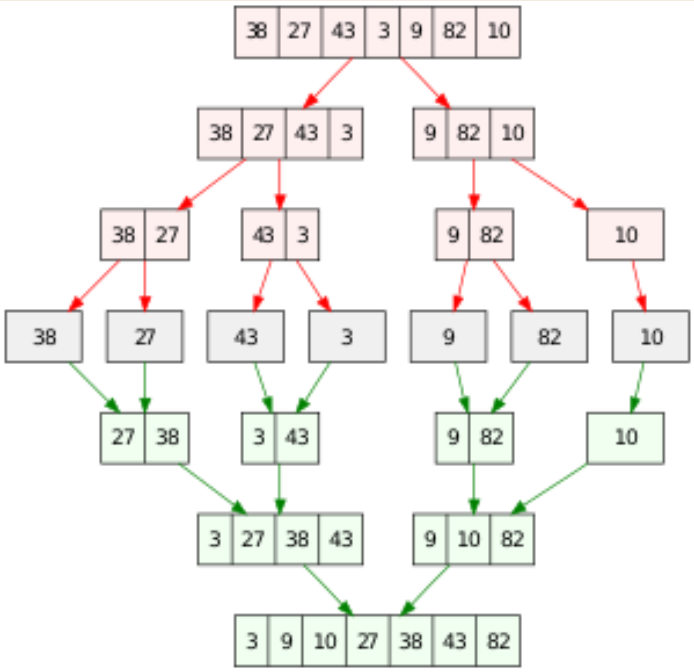


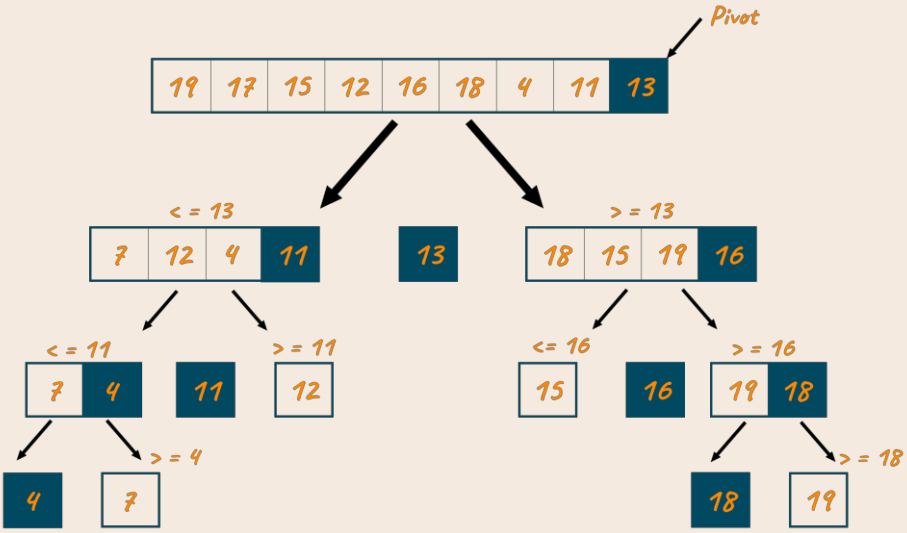
Insertion Sort

The first value in the data is considered sorted and the rest of the data is unsorted. Then it starts to compare the unsorted data values to the sorted data value if a value is smaller than the sorted then it is inserted in the sorted part. This goes on until the data is organized

Merge Sort

Here the data splits until the size of the data becomes one. Then the one data values are compared to each other and arranged in ascending order. after all the data is arranged it all merges back into one group of arranged data.



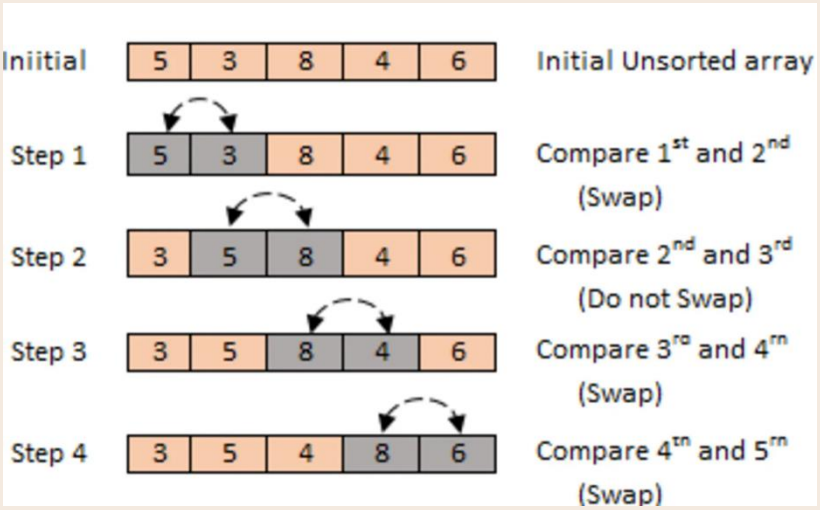


Quick Sort

Chooses a value as a pivot (the value to compare to) then it spits the data into two groups. Group1 is less than the pivot and group 2 is more than the pivot. Then we split both groups, in the same way, choosing a pivot in each group and so on until all the data is organized in ascending order.

Bubble sort

The simplest sorting algorithm works by repeatedly swapping the adjacent elements if they are in the wrong order.



Which sorting algorithm to use?

Sort Type	Advantage	Disadvantage
Selection	• Preforms better on small lists • No Additional storage is required	• Lack of efficiency when dealing with large lists
Bubble	• Easy to implement	• Does not work on lists containing a large number of values
Insertion	• Preforms better on small lists • Minimal space required	• Not efficient for huge lists
Merge	• Used for any data size	• Requires extra space
Quick	• Best storing • Preforms well with large lists	• Preforms less when the pivot is the smallest or largest value

Sorting Scenarios

	Best case	Average case	Worst case
Selection sort	N^2	N^2	N^2
Insertion sort	N	N^2	N^2
Merge sort	$Nlog(N)$	$Nlog(N)$	$Nlog(N)$
Quick sort	$Nlog(N)$	$Nlog(N)$	N^2

Our Program's Algorithm

```
Algorithm SelectionSort (Array  ArrayCopy1)
        Pre:  ArrayCopy1 is array to be sorted
             Post: ArrayCopy1 will be sorted ascendengly
                  return :The array ArrayCopy1 will be returned
            ordered

 for i=0 to ArrayCopy1.length-1 loop
  index=i
       for j=i+1 to ArrayCopy1.length-1 loop
            if ArrayCopy1[j] < ArrayCopy1[index] then

           index=j
              end if
                   end for j

    temp=ArrayCopy1[index]
        ArrayCopy1[index]=ArrayCopy1[i]
            ArrayCopy1[i]=temp

          end for i

end algorithm SelectionSort
```

```
Algorithm InsertionSort(Array ArrayCopy2)

 Pre : ArrayCopy2 is an array to be sorted
        Post : ArrayCopy2 Will Be Sorted
                Return : ArrayCopy2 Will Be Sorted And Returned

 for i=1 to ArrayCopy2.length-1 loop

  currentValue= ArrayCopy2[i]
        j=i-1

    loop while j>=0  and ArrayCopy2[j] > CurrentValue do

   ArrayCopy2[j+1] = ArrayCopy2[j]
      j=j-1

    end while loop

    ArrayCopy2[j+1]=CurrentValue
       end for i
             return ArrayCopy2

end algorithm Insertion Sort
```

```
Algorithm Merge ( Array ArrayCopy3 , leftHalf , rightHalf )

 leftSize = leftHalf.length

        rightSize= rightHalf.length

i=0
       j=0
              k=0

 while i < leftSize and  j<rightSize do

        if leftHalf[i]<= rightHalf[j]  then

              ArrayCopy3[k] = leftHalf[i]

                     i++

   else

       ArrayCopy3[k] = rightHalf[j]
              j = j+1

   end if

        k=k+1

end while

while  i<leftSize do
```

```
    ArrayCopy3[k] = leftHalf[i]
    i=i+1
      k=k+1
end while
while  j < rightSize  do
    ArrayCopy3[k] = rightHalf[j]
    k=k+1
       j=j+1
end while
end Algorithm
```

```
Algorithm MergeSort (Array ArrayCopy3)

  Pre Pre ArrayCopy3 is the array we want to sort

      Post Post ArrayCopy3 will be sorted

           return  ArrayCopy3 ordered

  CopyLength = ArrayCopy3.length
      Mid = CopyLength div 2

   if CopyLength<2 then

     return ArrayCopy3

   end if

   Array leftHalf = new int [Mid]
      Array rightHalf = new int [CopyLength - Mid]

   for i= 0 to Mid-1 loop

      leftHalf[i]=ArrayCopy3[i]

   end for

  for j=Mid  to  j<CopyLength  loop

     rightHalf[j-Mid] =ArrayCopy3[j]

  end for
```

call MergeSort(rightHalf)
call MergeSort(leftHalf)
call Merge (ArrayCopy3,leftHalf,rightHalf)

return ArrayCopy3

End Algorithm

```
Algorithm QuickSort(Array ArrayCopy4)

Pre ArrayCopy4 is the array we want to sort
      Post ArrayCopy4 will be sorted

call QuickSort(ArrayCopy4,0,ArrayCopy4.length-1)

   end Algorithm QuickSort



Algorithm QuickSort(Array ArrayCopy4, low , high)

  if low>= high then

    return ArrayCopy4

       randomPivot = choose pivot using Random class to choose
      random pivot
       pivot=ArrayCopy4[randomPivot]
            call swap (ArrayCopy4, randomPivot, high)

  leftPointer= call dividing (ArrayCopy4,low,high,pivot)
      call QuickSort(ArrayCopy4 , low , LeftPointer-1 )
           call QuickSort(ArrayCopy4, leftPointer+1 , high)

   return ArrayCopy4)

End Algorithm QuickSort
```

```
Algorithm Dividing(Array ArrayCoy4 , low , high , pivot)

 leftPointer=lowg
       rightPointer=high-1

 while leftPointer<rightPointer do
       while ArrayCopy4[leftPointer]<= pivot and leftPointer <
       rightPointer do

       leftPointer= leftPointer+1
    end while

    while ArrayCopy4[rightPointer]>= pivot and leftPointer<
rightPointer do

       rightPointer= rightPointer-1
             end while

   call swap (ArrayCopy4 , leftPointer , rightPointer)

 end while

if ArrayCopy4[leftPointer]>ArrayCopy4[high] then
   call swap(ArrayCopy4 , leftPointer , high)

else
        leftPointer = high

end if
       return leftPointer
             end Algorithm
```

Algorithm Swap(Array ArrayCopy4 , index1 , index2)

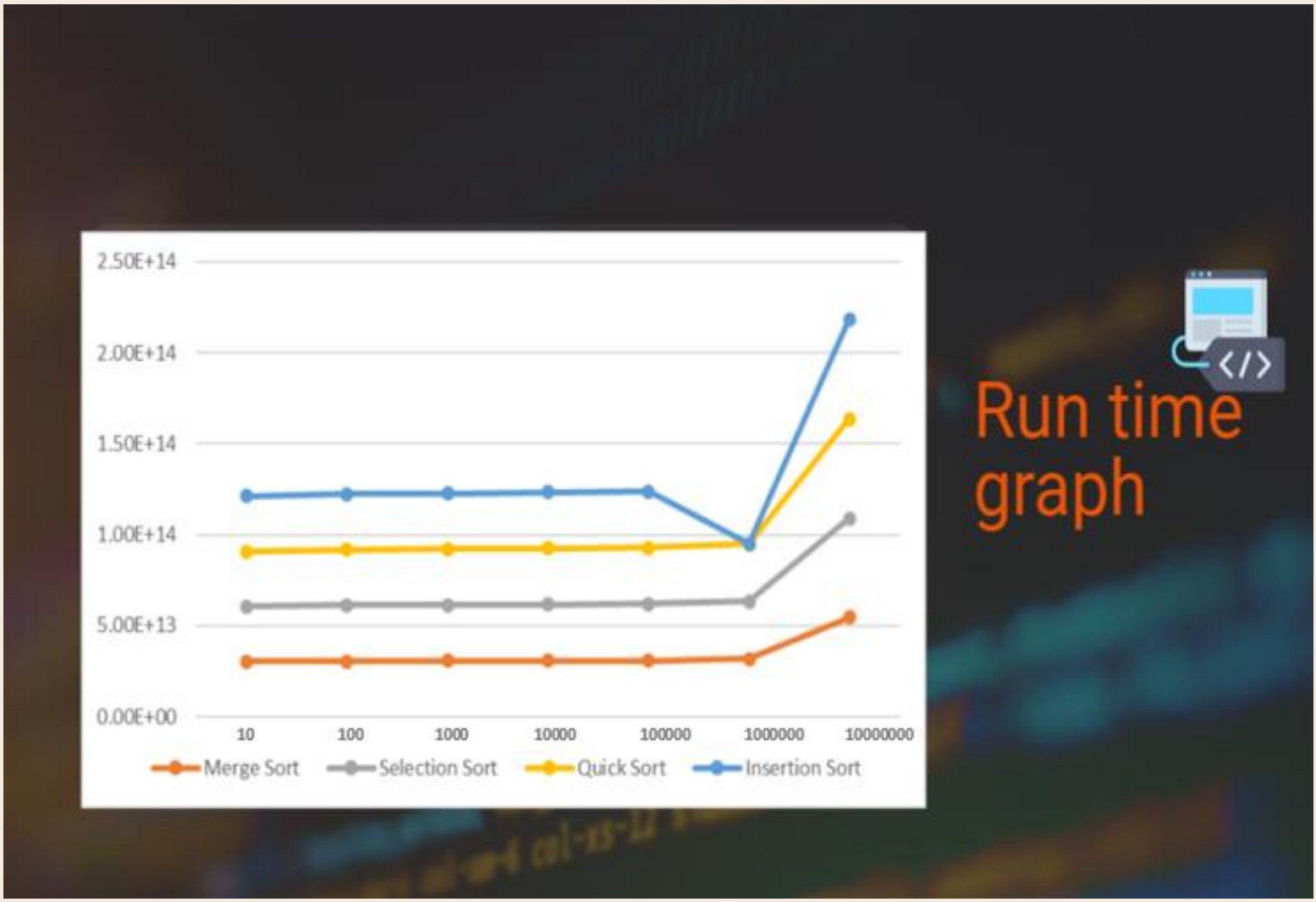
temp=ArrayCopy4[index1]
ArrayCopy4[index1]= ArrayCopy4[index2]
ArrayCopy4[index2]=temp

End Algorithm

Big O of N

Sorting Algorithm	Big O (N)
Merge	O(n log n)
Quick	O(n log n)
Insertion	$O(n^2)$
Selection	$O(n^2)$

Graph


2.50E+14
2.00E+14
1.50E+14
1.00E+14
5.00E+13
0.00E+00
10
100
1000
10000
100000
1000000
10000000
Merge Sort
Selection Sort
Quick Sort
Insertion Sort
Run time graph

References

https://www.geeksforgeeks.org/array-copy-in-java/

https://youtu.be/N2vUdSz1Y_w

https://www.youtube.com/watch?v=EwjnF7rFLns&ab_channel=BroCode

Insertion Sort Algorithm in Java - Full Tutorial With Source - YouTube

https://www.youtube.com/watch?v=bOk35XmHPKs&ab_channel=CodingwithJohn

https://www.youtube.com/watch?v=h8eyY7dIiN4&ab_channel=CodingwithJohn

https://docs.oracle.com/cd/B14099_19/bi.1012/b13915/sorting.htm

https://www.youtube.com/watch?v=cqh8nQwuKNE&ab_channel=JoeJames

https://www.youtube.com/watch?v=lCDZ0IprFw4&ab_channel=JoeJames

https://www.geeksforgeeks.org/bubble-sort/

https://www.geeksforgeeks.org/time-complexities-of-all-sorting-algorithms/

https://iq.opengenus.org/basic-sorting-algorithms-interview/