UNIVERSITY COLLEGE OF APPLIED SCIENCES

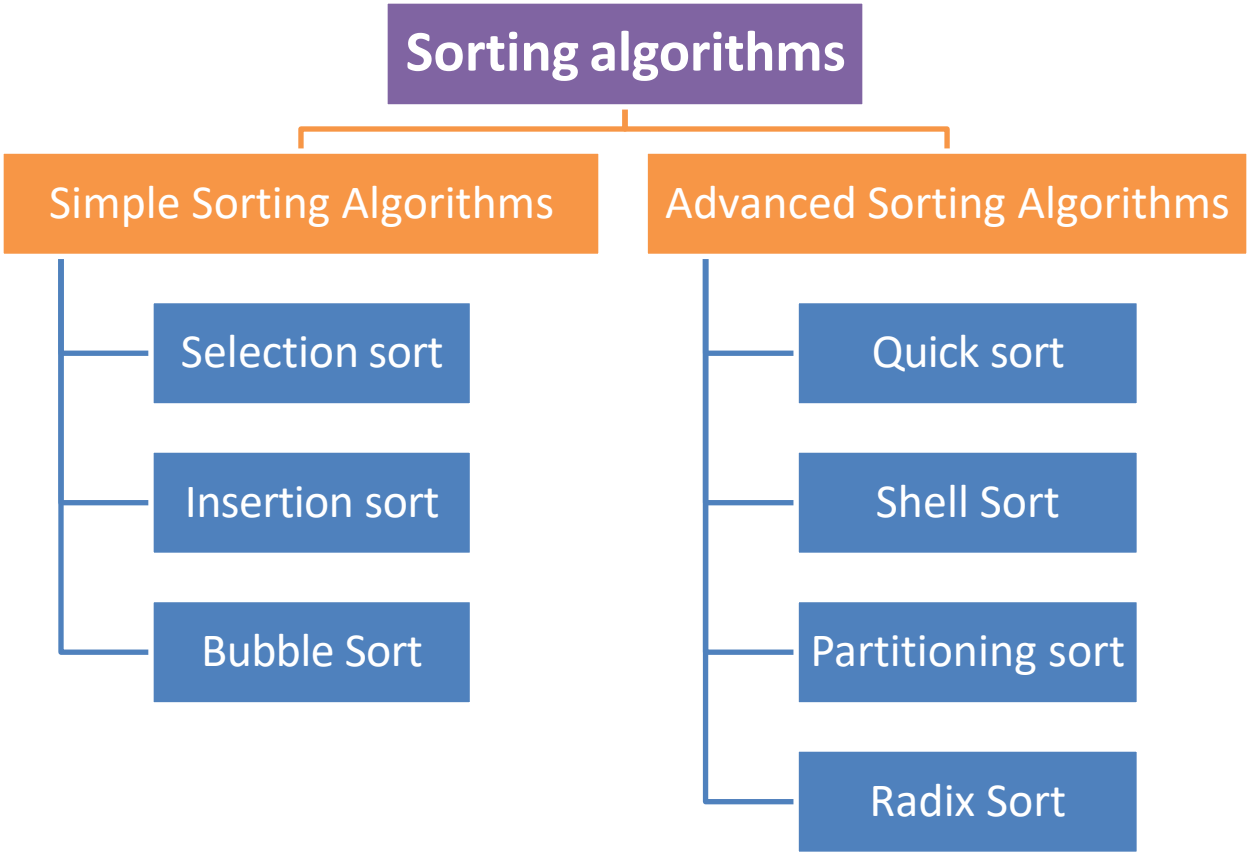
ALGORITHMS AND DATA STRUCTURES

SORTING ALGORITHMS

Sorting algorithms

There are many algorithms that perform various operations on arrays. However, one of the most common tasks is sorting an array to arrange its elements in the correct order, either ascending or descending. The topic of sorting algorithms involves many approaches, including selection sort, insertion sort, bubble sort, and quicksort, which will be explained in detail in this part of the chapter

Sorting algorithms



Sorting algorithms

Selection Sort

1. Selection sort

It is one of the simplest sorting algorithms. The algorithm divides the array into two parts, namely sorted and unsorted. In the following iterations, the algorithm finds the smallest element in the unsorted part and exchanges it with the first element in the unsorted part. It sounds very simple, doesn't it?

How selection sort works?

Lets consider the following array as an example :
Arr[]={64,25,12,22,11}

First pass:

- For the first position in the sorted array, the whole array is traversed from index 0 to 4 sequentially. The first position where 64 is stored presently, after traversing whole array it is clear that 11 is the lowest value.

64	25	12	22	11

- Thus, replace 64 with 11. After one iteration 11, which happens to be the least value in the array, tends to appear in the first position of the sorted list.

11	25	12	22	64

How selection sort works?

Second Pass:

For the second position, where 25 is present, again traverse the rest of the array in a sequential manner.

11	25	12	22	64

After traversing, we found that 12 is the second lowest value in the array and it should appear at the second place in the array, thus swap these values.

11	12	25	22	64

How selection sort works?

Third Pass:

Now, for third place, where 25 is present again traverse the rest of the array and find the third least value present in the array.

While traversing, 22 came out to be the third least value and it should appear at the third place in the array, thus swap 22 with element present at third position.

Selection Sort : Code Example Java

```
public void selectionSort()
   {
   int out, in, min;

   for(out=0; out<nElems-1; out++)    // outer loop
      {
      min = out;                      // minimum
      for(in=out+1; in<nElems; in++) // inner loop
         if(a[in] < a[min] )          // if min greater,
             min = in;                // we have a new min
      swap(out, min);                 // swap them
      }  // end for(out)
   }  // end selectionSort()
```

For complete code return to text book page 93

Time Complexity: The time complexity of Selection Sort is $O(N^2)$ as there are two nested loops.

Insertion Sort

Insertion Sort:

The insertion sort is another algorithm that makes it possible to sort a single-dimensional array in a simple way. Similarly, as in the case of the selection sort, the array is divided into two parts, namely sorted and unsorted. However, at the beginning, the first element is included in the sorted part. In each iteration, the algorithm takes the first element from the unsorted part and places it in a suitable location within the sorted part, to leave the sorted part in the correct order. Such operations are repeated until the unsorted part is empty.

How Insertion sort works?

Consider an example: arr [ ]={12,11,13,5,6}

12	11	13	5	6

First Pass:

Initially, the first two elements of the array are compared in insertion sort.

12	11	13	5	6

Here, 12 is greater than 11 hence they are not in the ascending order and 12 is not at its correct position. Thus, swap 11 and .12

So, for now 11 is stored in a sorted sub-array.

11	12	13	5	6

How Insertion sort works?

Second Pass:

Now, move to the next two elements and compare them

11	12	13	5	6

Here, 13 is greater than 12, thus both elements seems to be in ascending order, hence, no swapping will occur. 12 also stored in a sorted sub-array along with 11

Third Pass:

Now, two elements are present in the sorted sub-array which are 11 and 12
Moving forward to the next two elements which are 13 and 5

How Insertion sort works?

11	12	13	5	6

Both 5 and 13 are not present at their correct place so swap them

11	12	5	13	6

After swapping, elements 12 and 5 are not sorted, thus swap again

11	5	12	13	6

Here, again 11 and 5 are not sorted, hence swap again

5	11	12	13	6

Here, 5 is at its correct position

How Insertion sort works?

Fourth Pass:

Now, the elements which are present in the sorted sub-array are 5, 11 and 12

Moving to the next two elements 13 and 6

5	11	12	13	6

Clearly, they are not sorted, thus perform swap between both

5	11	12	6	13

Now, 6 is smaller than 12, hence, swap again

5	11	6	12	13

Here, also swapping makes 11 and 6 unsorted hence, swap again

5	6	11	12	13

Insertion Sort: Code Example java

```
public void insertionSort()
   {
   int in, out;

   for(out=1; out<nElems; out++)     // out is dividing line
      {
      long temp = a[out];         // remove marked item
      in = out;                        // start shifts at out
      while(in>0 && a[in-1] >= temp) // until one is smaller,
         {
         a[in] = a[in-1];              // shift item right,
         --in;                         // go left one position
         }
      a[in] = temp;                    // insert marked item
      }  // end for
   }  // end insertionSort()
```

For complete code return to text book page 101

Time Complexity: $O(N^2)$

Bubble Sort

Bubble Sort

Its way of operation is very simple, because the algorithm just iterates through the array and compares adjacent elements. If they are located in an incorrect order, they are swapped. It sounds very easy, but the algorithm is not very efficient and its usage with large collections could cause performance-related problems.

How does Bubble Sort Work?

Input: arr []={ 5,1,4,2,8 }

First Pass:

•Bubble sort starts with very first two elements, comparing them to check which one is greater.

•Here, algorithm compares the first two elements, and swaps since 5 > 1.

{ **5,1**,4,2,8 } -> ={ **1,5**,4,2,8 } Swap since 5 > 1

{ 1,**5,4**,2,8 } -> ={ 1,**4,5**,2,8 } Swap since 5 > 4

{ 1,4,**5,2**,8 } -> ={1,4,**2,5**,8 } Swap since 5 > 2

{1,4,2,**5,8** } -> ={1,4,2,**5,8** } No Swap

How does Bubble Sort Work?

Second Pass:

- Now, during second iteration it should look like this:

{**1,4**,2,5,8 } -> ={**1,4**,2,5,8 } No Swap

{1,**4,2**,5,8 } -> ={1,**2,4**,5,8 } Swap since 4 > 2

{1,2,**4,5**,8 } -> ={1,2,**4,5**,8 } No Swap

How does Bubble Sort Work?

Third Pass:

- Now, during Third iteration it should look like this:

{**1,2**,4,5,8 } -> ={**1,2**,4,5,8 } No Swap

{1,**2,4**,5,8 } -> ={1,**2,4**,5,8} No Swap

How does Bubble Sort Work?

Fourth Pass:

- Now, during Fourth iteration it should look like this:

{**1,2**,4,5,8 } -> ={**1,2**,4,5,8 } No Swap

Bubble Sort : Example Code Java

```
public void bubbleSort()
 {
 int out, in;

 for(out=nElems-1; out>1; out--)    // outer loop (backward)
    for(in=0; in<out; in++)         // inner loop (forward)
       if( a[in] > a[in+1] )        // out of order?
          swap(in, in+1);           // swap them
 }  // end bubbleSort()
```

For complete code return to text book page 85

Time Complexity: $O(N^2)$

Sorting algorithms

Quicksort

To be discussed later