Chapter 2

Array

Array

By: Sayed Hassan Adelyar

Array

Array

- **Most commonly** used data storage structure.
- A **set** of **elements** stored in computer memory.
- All the elements have the **same name** & **type** and are differentiate by an **index**.

Array

```
public class student {
    int idno;
    String name;
    int score;
public student (int id, String stname, int sc) {
        idno = id;
        name = stname;
        score = sc;
    }

}
```

Array

```
class arrayapp {

    public static void main (String [] args){
    student stscore[] = new student [12];
        int i;
      student st1 = new student(22, "safi", 80);
      student st2 = new student(33, "zabi", 75);
      student st3 = new student(44, "khan", 90);
      student st4 = new student(55, "wali", 70);

           stscore[0] = st1;
           stscore[1] = st2;
           stscore[2] = st3;
           stscore[3] = st4;

      for(i=0; i<4; i++) {
          System.out.println(stscore[i].idno);
          System.out.println(stscore[i].name);
          System.out.println(stscore[i].score);
       }

    }

}
```

Searching in Array

Array

- Two search Algorithm are available:
  - Linear Search,
  - Binary Search.

Linear Search Algorithm

Array

```
public student find(String searchname)
{
    int j;
    for(j=0; j<nelems; j++)
        if( a[j] = searchname)
            break;
    if( j == nelems)
        return null;
    else
        return a[j];
}
```

Binary Search Algorithm

Array


```
public boolean find(int searchkey) {
   int lowerBound = 0;
   int upperBound = n-1;
   int middle=0;
   while(true) {
      middle = (lowerBound +     upperBound)
     / 2;
      if (student[middle] == searchkey)
         return true;
      else if(lowerBound > upperBound)
         return false;
```

```
else {
      if(student[middle] <
searchkey)
     lowerBound = middle + 1;
      else
    upperBound = middle - 1;
         }
        }
    }
```

Sorting

Array

- Sorting is, without doubt, the most **fundamental algorithmic** problem.

1. **Supposedly**, **25**% of all **CPU** cycles are spent sorting
2. Sorting is fundamental to most other algorithmic problems, for example **binary search**.
3. Many **different approaches** lead to **useful sorting algorithms**, and these ideas can be used to solve many other problems.

Array

- **Applications of Sorting**
  - **Speeding** up **searching**.
  - **Given n numbers**, find the **pair** which is closest to each other.
  - Given a set of n items, are they all **unique** or are there any **duplicates**?
  - **Frequency distribution** - Given a set of n items, which element occurs the largest number of times?
  - What is the **kth largest** item in the set?

Bubble Sort

Array

- **Start** at the **left** end of the line and **compare** the elements in position **0 & 1**.
- If the one on the **left** is **bigger**, you **swap** them.
- **Move** over **one position** and compare the elements in positions **1 and 2**.
- If the one on the **left** is **bigger**, you **swap** them.
- **Continue** down the line this way until you **reach** the **right end**.
- **Go back** and **start** another **pass** from the **left** end of line, go toward the **right**, **comparing** and **swapping**. This time you can **stop one item** short of the end of the line, at **position N-2**, because you know the last position, at **N-1**, **already** contains the **biggest** item.
- **Continue** this **process** until all the elements are in order.

Array

```
public void bubblesort()
{
 int out, in;
 for(out=n-1; out>0; out--)
   for(in=0; in<out; in++)
     if( student[in] > student[in+1] )
         swap(in, in+1);
}
```

```
public void swap (int one,
    int two)
{
    int temp = student[one];
    student[one] =
    student[two];
    student[two] = temp;
}
```

Bubble Sort

Selection Sort

Array

- The **most natural** and easiest sorting algorithm.
- **Repeatedly find** the **smallest** element, move it to the **front**.
- The selection sort **improves** on the bubble sort by **reducing** the number of **swaps** necessary from **O(N^2)** to **O(N)**.
- Unfortunately, the number of **comparisons** remains **O(N^2)**.
- However, the selection sort can still offer a **significant improvement** for **large records** that must be physically moved around in memory, causing the **swap time** to be much more important than the comparison time.

Array

```
public void selectionsort() {
    int out, in, min;
    for(out=0; out<n-1; out++) {
        min = out;
        for(in=out+1; in<n; in++)
            if(student[in] < student [min] )
                min=in;
                swap(out, min);
    }
}
```

Selection Sort

Quick Sort

Array

- The **most popular** sorting algorithm.
- In the **majority** of **situations**, it is the **fastest**, operating in **O(N*logN)** time.
- Quick sort was **discovered** by **C.A.R**. **Hoare** in 1962.
- Operates by **partitioning** an array into two sub-arrays and then calling itself recursively to quick sort each of these sub-arrays.

Array

Quick Sort

```
public void quicksort()
{
    recquicksort(0, n-1);
}
public void recquicksort(int left, int right)
{
    if(right-left <= 0)
        return;
    else
    {
        long pivot = a[right];

        int partition = partitionit(left, right, pivot);
        recquicksort(left, partition-1);
        recquicksort(partition+1, right);
    }
}
```

Array

```
public int partitionit(int left, int right, long pivot)
{
    int leftptr = left -1;
    int rightptr = right;
    while(true) {
        while(a[++leftptr] < pivot)
            ;
        while(rightptr > 0 && a[--rightptr] > pivot)
            ;
        if(leftptr >= rightptr)
            break;
        else
            swap(leftptr, rightptr);
    }
    swap(leftptr, right);
        return leftptr;
    }
```

```
public void swap(int dex1,
int dex2)
    {
        long temp = a[dex1];
        a[dex1] = a[dex2];
        a[dex2] = temp;
    }
```

Complete Java Program for Quick Sort

Array

```
import java.util.Random;
class starrayapp
{
    public static void main(String[] args)
    {
        int maxsize = 100;
        starray stlist;
        stlist = new starray(maxsize);
        int item;
        Random generator2 = new Random();
        for( int i= 1; i<25; i++){
            item = generator2.nextInt(100) + 0;
        stlist.insert(item);
        }
        System.out.println("List of items before sorting");
        stlist.display();
        stlist.quicksort();
        System.out.println("List of items after sorting");
        stlist.display();
    }
}
```

Array

```
class starray
{
private int[] student;
private int n;

public starray (int max)
{
    student = new int[max];
    n = 0;
}
```

Array

```
public void insert(int value)
{
    student[n] = value;
    n++;
}

public void quicksort()
{
    recquicksort(0, n-1);
}
public void recquicksort(int left, int right)
{
    if(right-left <= 0)
        return;
    else
    {
        long pivot = student[right];

        int partition = partitionit(left, right, pivot);
        recquicksort(left, partition-1);
        recquicksort(partition+1, right);
    }
}
```

Array

```
public int partitionit(int left, int right, long pivot)
{
    int leftptr = left -1;
    int rightptr = right;
    while(true) {
        while(student[++leftptr] < pivot)
            ;
        while(rightptr > 0 && student[--rightptr] > pivot)
            ;
        if(leftptr >= rightptr)
            break;
        else
            swap(leftptr, rightptr);
    }
    swap(leftptr, right);
        return leftptr;
    }
```

Array

```
public void swap(int dex1, int dex2)
    {
        int temp = student[dex1];
        student[dex1] = student[dex2];
        student[dex2] = temp;
    }

public void display()
{
    for(int j=0; j<n; j++)
        System.out.print(student [j] + "  ");
    System.out.println("  ");
}
}
```