Array ADT

- Array ADT (Abstract Data Type)
- The representation of data is define by the compiler itself , However the operations on the data must be given by the program . The combination of theses 2 on an array is called Array ADT

- Some possible operations on array are :

Display( )

Add(x) / Append(x)

Insert(index , x)

Delete(index)

Search(x)

Get(index)

Set(index , x)

Max( ) / Min( )

Reverse( )

Shift( ) / Rotate( )

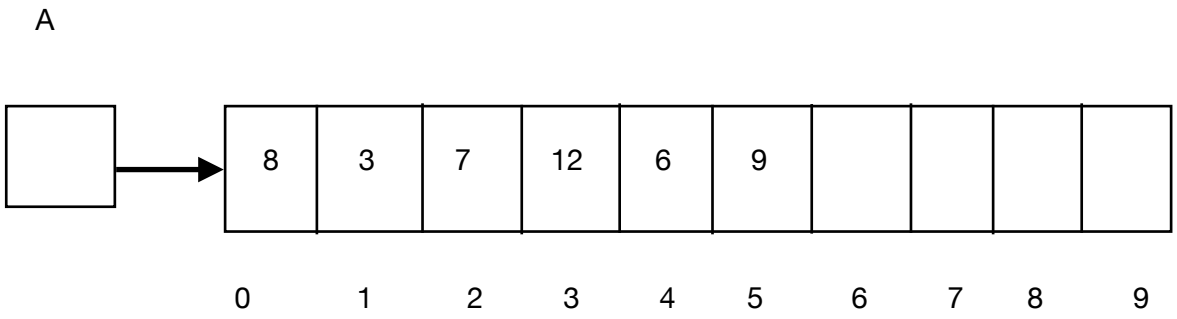
- The representation of array data require 3 thing

1. Array Space
2. Size
3. length(no.of element)

Inserting in an Array

- First consider the following array as an example

Size = 10
Length = 6



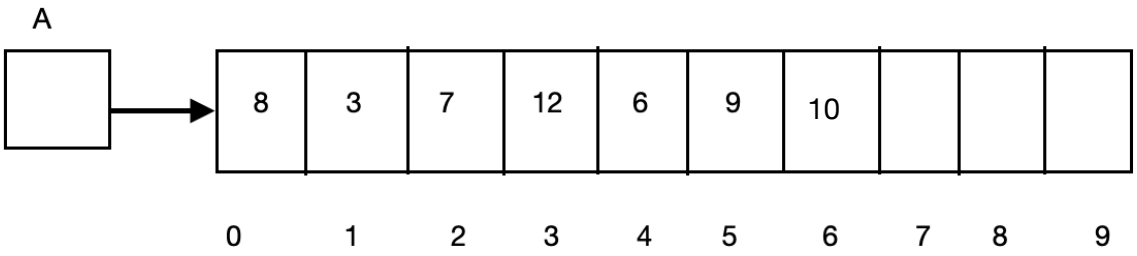
- For **displaying** the elements

Syntax

```
for( i= 0;  i < length; i++ )
{
        print( A[ i ] )
}
```

- **Add(x) / Append (x)** : adding an element at the end of an array that is adding in the next free space

Size = 10
Length = 7



Syntax

```
A[ Length ] = x ;
Length++;
```

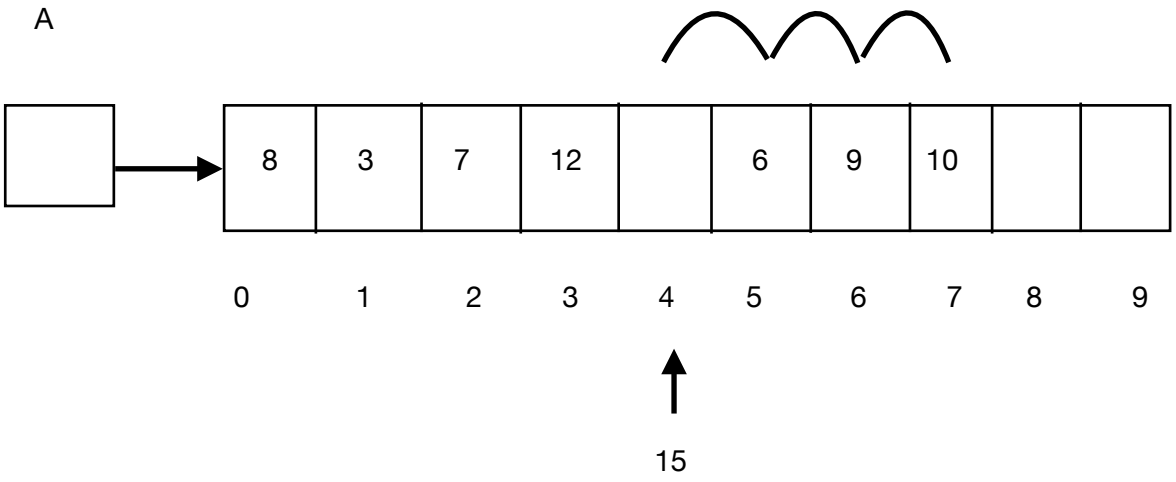
- **Insert(index, x)** : It takes index and element, meaning to insert an element in a given index

- If the space is free the value will be inserted automatically , but if the space is already taken by another element it must be moved to next space in order to create space for the new insert value

Syntax : Insert(4, 15)

```
for( i = Length; i > index ; i - -)
{
        A[ i ] = A[ i -1] ;
}
A[ index ] = x ;
Length ++;
```

A



Inserting and Appending in a Array

```c
#include<stdio.h>
struct Array
{
    int A[10];
    int size;
    int length;
};

    void Display(struct Array arr)
    {
        int i;
        printf("\nElements are\n");
        for(i=0;i<arr.length;i++)
            printf("%d ",arr.A[i]);
    }
    void Append(struct Array *arr,int x)
    {
        if(arr->length<arr->size)
            arr->A[arr->length++]=x;
    }
    void Insert(struct Array *arr,int index,int x)
   {
    int i;

    if(index>=0 && index <=arr->length)
    {
        for(i=arr->length;i>index;i--)
            arr->A[i]=arr->A[i-1];
        arr->A[index]=x;
        arr->length++;
    }
   }

int main()
{

struct Array arr1={{2,3,4,5,6},10,5};
Append(&arr1,10);
Insert(&arr1,0,12);
Display(arr1);
    return 0;
}
```

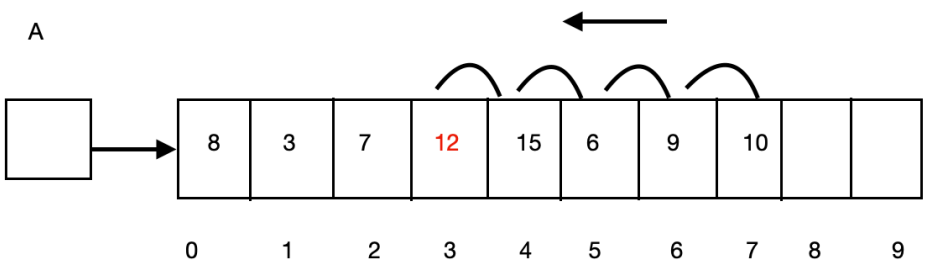
Deleting from Array

- Removing an element from an array is called deleting
- After deleting an element the space must not be empty in an array so shift the bits accordingly
- The index should not be beyond the array

Syntax : Delete(3)

```
x = A[ index ]
for(  i = index ; i < length - 1 ; i++ )
{
        A[ i ] = A[ i+1] ;
}
```

Size = 10
Length = 8



Deleting from Array

```c
#include<stdio.h>
struct Array
{
    int A[10];
    int size;
    int length;
};
    void Display(struct Array arr)
    {
        int i;
        printf("\nElements are\n");
        for(i=0;i<arr.length;i++)
            printf("%d ",arr.A[i]);
    }

int Delete(struct Array *arr,int index)
{
    int x=0;
    int i;
    if(index>=0 && index<arr->length)
    {
        x=arr->A[index];
        for(i=index;i<arr->length-1;i++)
            arr->A[i]=arr->A[i+1];
        arr->length--;
        return x;
    }
    return 0;
}

int main()
{

    struct Array arr1={{2,3,4,5,6},10,5};
    printf("%d",Delete(&arr1,0));
    Display(arr1);
    return 0;
}
```

Ds

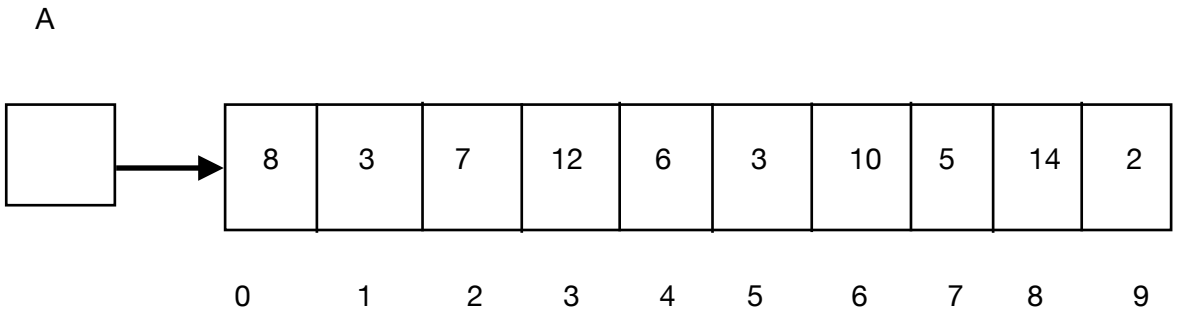
Linear Search

- They are 2 search method in an array

I. Linear search
II. Binary search

- Linear search :

Size = 10
Length = 10



Key = 5 (successful search)
Key = 12 (unsuccessful search)

- All the elements must be unique here
- The value you are searching is called key, In linear search we search the key element one by one linearly
- We search the element by comparing it with the key value

- The result of the search is the location of the element where its present (index number) , it is very useful in accessing the element in the list

- If the element is not found throughout the list that means it is not present in the list therefore search is unsuccessful

Syntax :

```
for( i = 0;  i < length ; i++ )
{
        if( key == A[ i ] )
        return i;                    //if search is successful it ends here
}

return -1;                           // if search unsuccessful returns -1
```

Improving Linear Search

- When you are searching for a key element there is a possibility that you are searching the same element again
- To improve the speed of comparison , you can move a key element repeatedly search one step forward this method is called transposition

Syntax :

```
for( i = 0; i < length ; i++ )
        {
                if( key == A[ i ] )
                {
                        swap( A[i], A[i-1]);
                        return i-1;
                }
        }
```

- The second method is you can directly swap the key element to the first element this process is called move to head . The next search for the same element becomes faster.

```
for( i = 0; i < length ; i++ )
        {
                if( key == A[ i ] )
                {
                        swap( A[i], A[0]);
                        return 0;
                }
        }
```

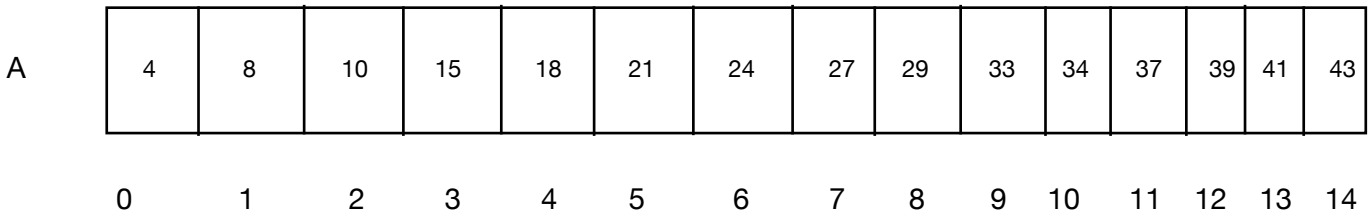
Searching in a Array

```c
#include<stdio.h>
struct Array
{
    int A[10];
    int size;
    int length;
};
    void Display(struct Array arr)
    {
        int i;
        printf("\nElements are\n");
        for(i=0;i<arr.length;i++)
            printf("%d ",arr.A[i]);
    }
 void swap(int *x,int *y)
 {
     int temp=*x;
     *x=*y;
     *y=temp;
 }

int LinearSearch(struct Array *arr,int key)
{
    int i;
    for(i=0;i<arr->length;i++)
    {
        if(key==arr->A[i])
        {
            swap(&arr->A[i],&arr->A[0]);
            return i;
        }
    }
    return -1;
}


int main()
{

    struct Array arr1={{2,23,14,5,6,9,8,12},10,8};
    printf("%d",LinearSearch(&arr1,14));
    Display(arr1);
    return 0;
}
```

Binary Search

- The condition for binary search is that the list of elements must be sorted

Example :



- The binary search will always search the element in the middle of the list and split it into 2 parts
- For performing binary search we need 3 index that is lower , higher , middle value

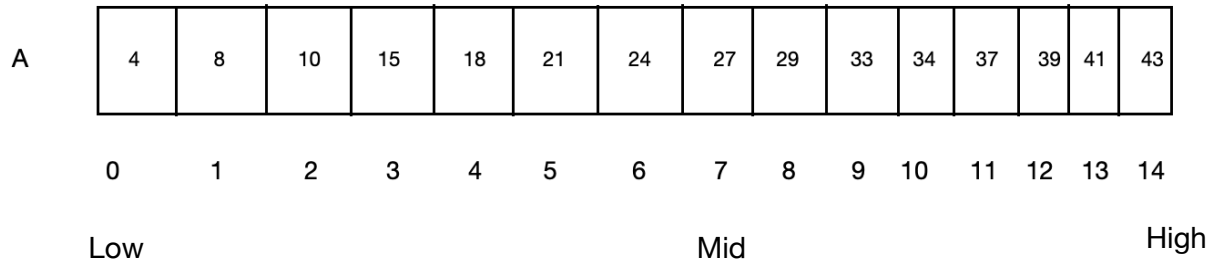
$$mid = [l + h / 2]$$

- Low will point at initial value that is index 0
- high will point at the end of the list
- Mid will point the the centre most value in a list

- If the number we are searching is greater than key element it will search the value o left hand side and if the value is lower than the key it will check in right hand side

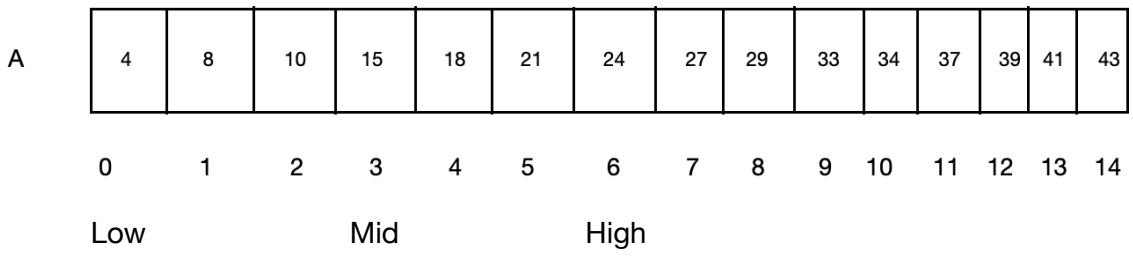
Example 1 :

Size = 15
Length = 15

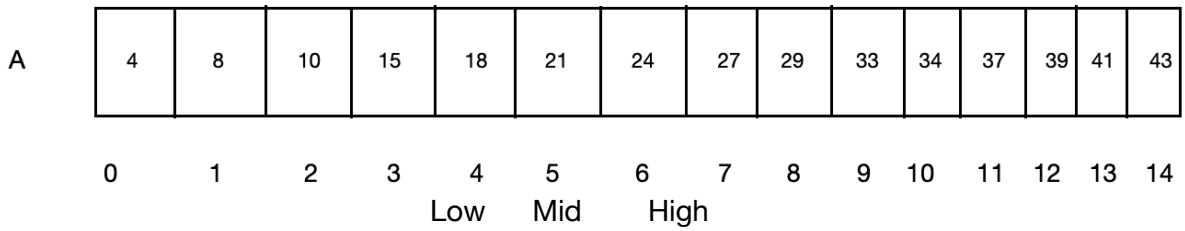


Suppose key = 18

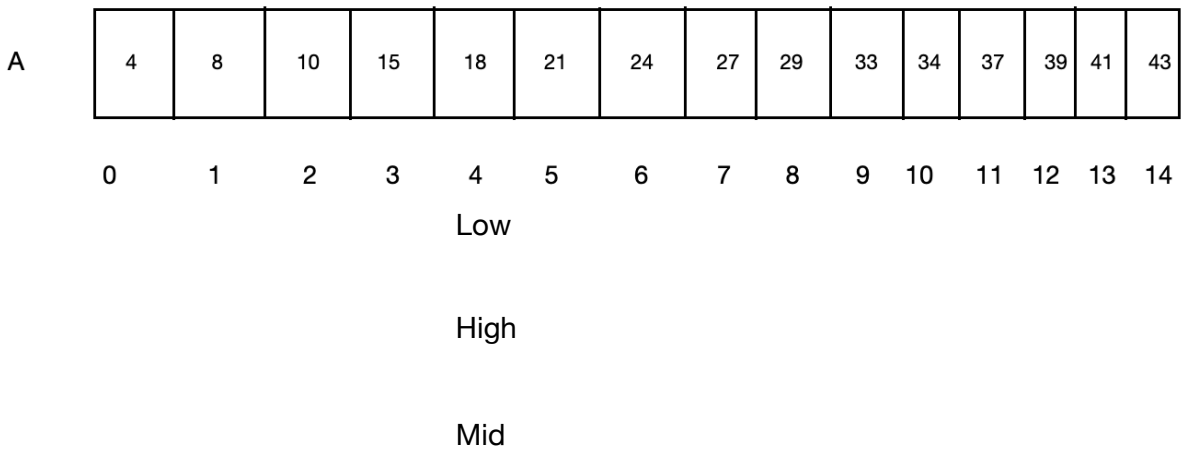
- Once the value we want is know the list is divides again to check the element in that half



- Once again the same procedure is repeated



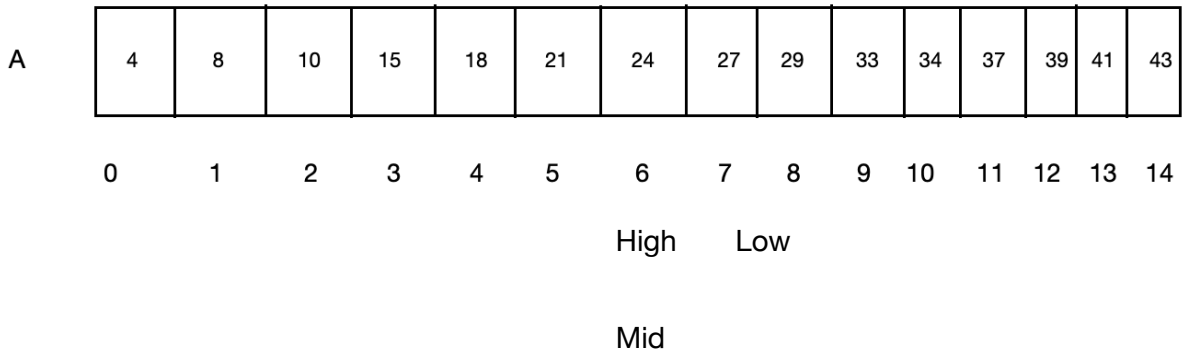
- The list value is getting reduced and every time it is getting divided by 2
- When the same steps is performed again all the values (low , high , mid) will point to the same number which will be the search value.



- Hence the search is successful

Example 2 :

Key = 25



Key = 25

L	H	mid
0	14	7
0	6	3
4	6	5
6	6	6
7	6	x

- When low became greater than high we stop the process it indicated that the element is not present in the list thus the search is unsuccessful

Binary Search Algorithm

- The algorithm for binary search is as follows

iterative procedure

```
Algorithm BinSearch(l,h,key)
{
     while(l<=h)
     {
          mid = [(l+h)/2];
          if(key==A[mid])
               return mid;
          else if (key<A[mid])
               h=mid-1;
          else
               l=mid+1;
     }
     return-1;
}
```

Recursive procedure

```
Algorithm RBinSearch(l,h,key)
{
    if(l<=h)
    {
        mid = [(l+h)/2];
        if(key==A[mid])
            return mid;
        else if (key<A[mid])
            return RBinSearch(l, mid-1, key);
        else
            return RBinSearch(mid+1, h, key);
    }
    return-1;
}
```

- Tail and loop recursive are similar
- If given option for both always go for loop recursive as its better than it because it uses stack

Binary Search in Array

```
#include<stdio.h>
struct Array
{
    int A[10];
    int size;
    int length;
};
    void Display(struct Array arr)
    {
        int i;
        printf("\nElements are\n");
        for(i=0;i<arr.length;i++)
            printf("%d ",arr.A[i]);
    }
 void swap(int *x,int *y)
 {
     int temp=*x;
     *x=*y;
     *y=temp;
 }

int BinarySearch(struct Array arr,int key)
{
    int l,mid,h;
    l=0;
    h=arr.length-1;
    while(l<=h)
    {
        mid=(l+h)/2;
        if(key==arr.A[mid])
            return mid;
        else if(key<arr.A[mid])
            h=mid-1;
        else
            l=mid+1;
    }
return -1;
}

int RBinSearch(int a[],int l,int h,int key)
{

    int mid=0;
    if(l<=h)
    {
```

```
        mid=(l+h)/2;
        if(key==a[mid])
            return mid;
        else if(key<a[mid])
            return RBinSearch(a,l,mid-1,key);
    }
    else
        return RBinSearch(a,mid+1,h,key);
return -1;
}

int main()
{

    struct Array arr1={{2,3,9,16,18,21,28,32,35},10,9};
    printf("%d",BinarySearch(arr1,16));
    Display(arr1);
    return 0;
}
```

Ds

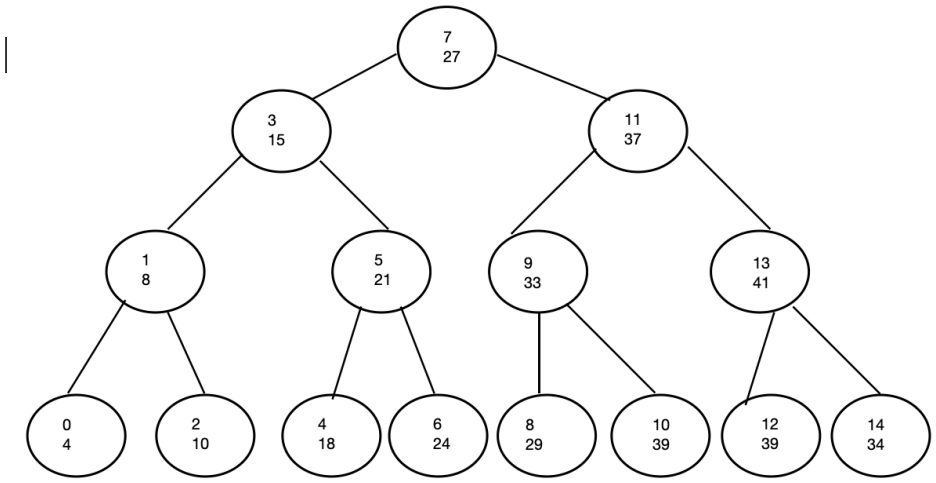
Analysis of Binary Search

Size = 15
Length =15

A

4	8	10	15	18	21	24	27	29	33	34	37	39	41	43
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

The tracing tree for the following list is as follows



- For successful search

The best case time for this problem is min - 0(1)
The worst case time for this problem is max - 0(logn)

- For unsuccessful search

Its always 0(log n) weather the call be recursive or iterative

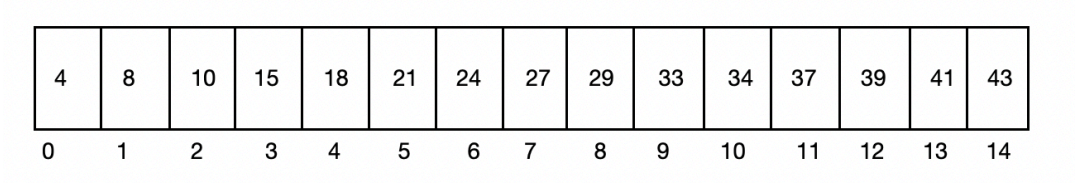
- The max n0 of calls this problem will be making is 4 and if the element is not the call will be 5

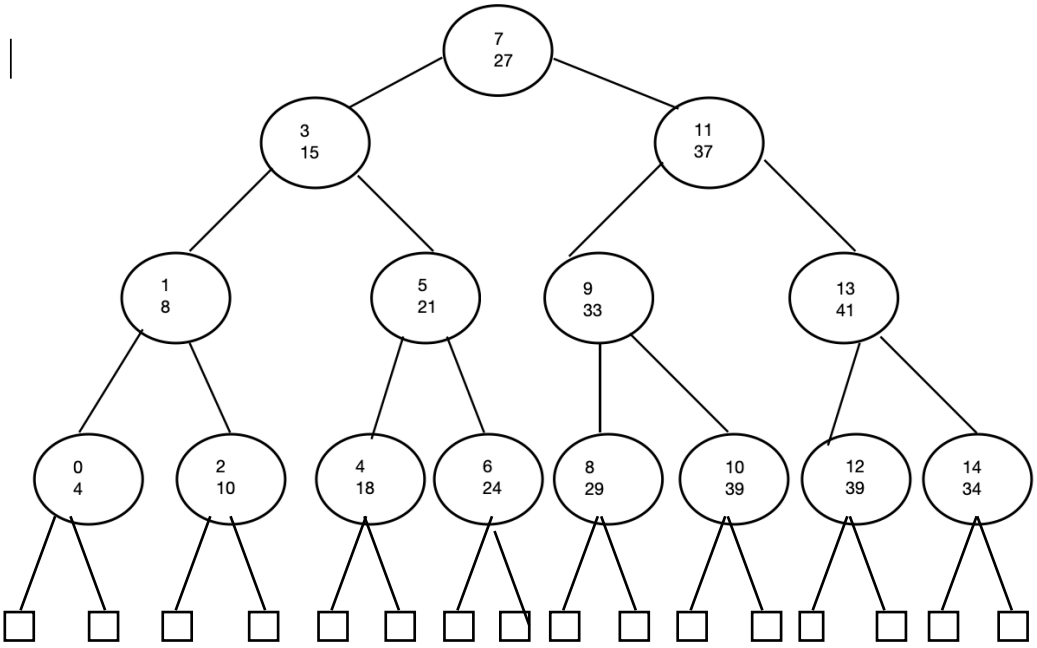
Ds

Average case analysis of binary search

Size = 15
Length = 15

A





- For finding the average case time we have to first consider the internal nodes , the no of comparison for internal nodes depends on the level of tracing tree the circular nodes **(I)** are the internal nodes which are successful

- The square nodes are the unsuccessful ones **(E)**

E = nlogn
E = I + 2n
e = I + 1

A_s (n) = 1 + I / n

A_u (n) = E / n + 1 = nlogn / n + 1

- The average unsuccessful time is logn

- The average successful time is also logn

Get Set Max Min on Array

```c
#include<stdio.h>
struct Array
{
    int A[10];
    int size;
    int length;
};
    void Display(struct Array arr)
    {
        int i;
        printf("\nElements are\n");
        for(i=0;i<arr.length;i++)
            printf("%d ",arr.A[i]);
    }
 void swap(int *x,int *y)
 {
     int temp=*x;
     *x=*y;
     *y=temp;
 }
int Get(struct Array arr,int index)
{
    if(index>=0 && index<arr.length)
        return arr.A[index];
    return -1;
}

void Set(struct Array *arr,int index,int x)
{
    if(index>=0 && index<arr->length)
    arr->A[index]=x;
}

int Max(struct Array arr)
{
    int max=arr.A[0];
    int i;
    for(i=1;i<arr.length;i++)
    {
        if(arr.A[i]>max)
            max=arr.A[i];
    }
    return max;
}
```

```c
int Min(struct Array arr)
{
    int min=arr.A[0];
    int i;
    for(i=1;i<arr.length;i++)
    {
        if(arr.A[i]<min)
            min=arr.A[i];
    }
    return min;
}

int Sum(struct Array arr)
{
    int s=0;
    int i;
    for(i=0;i<arr.length;i++)
        s+=arr.A[i];
    return s;
}
float Avg(struct Array arr)
{
    return (float)Sum(arr)/arr.length;
}
int main()
{

    struct Array arr1={{2,3,9,16,18,21,28,32,35},10,9};
    printf("%d",Sum(arr1));
    Display(arr1);
    return 0;
}
```

Reversing an Array

```c
#include<stdio.h>
#include<stdlib.h>
struct Array
{
    int A[10];
    int size;
    int length;
};
    void Display(struct Array arr)
    {
        int i;
        printf("\nElements are\n");
        for(i=0;i<arr.length;i++)
            printf("%d ",arr.A[i]);
    }
 void swap(int *x,int *y)
 {
     int temp=*x;
     *x=*y;
     *y=temp;
 }
void Reverse(struct Array *arr)
{
    int *B;
    int i,j;

    B=(int *)malloc(arr->length*sizeof(int));
    for(i=arr->length-1,j=0;i>=0;i--,j++)
        B[j]=arr->A[i];
    for(i=0;i<arr->length;i++)
        arr->A[i]=B[i];

}

void Reverse2(struct Array *arr)
{
    int i,j;
    for(i=0,j=arr->length-1;i<j;i++,j--)
    {
        swap(&arr->A[i],&arr->A[j]);
    }
}
int main()
{
```

```
    struct Array arr1={{2,3,9,16,18,21,28,32,35},10,9};
    Reverse(&arr1);
    Display(arr1);
    return 0;
}
```

Checking if Array is Sorted

```c
#include<stdio.h>
#include<stdlib.h>
struct Array
{
    int A[10];
    int size;
    int length;
};
    void Display(struct Array arr)
    {
        int i;
        printf("\nElements are\n");
        for(i=0;i<arr.length;i++)
            printf("%d ",arr.A[i]);
    }

int isSorted(struct Array arr)
{
    int i;
    for(i=0;i<arr.length-1;i++)
    {
        if(arr.A[i]>arr.A[i+1])
            return 0;
    }
    return 1;
}

int main()
{

    struct Array arr1={{2,3,9,16,18,21,28,32,35},10,9};
    printf("%d",isSorted(arr1));
    Display(arr1);
    return 0;
}
```

Merging 2 Arrays

```
struct Array
{
    int A[10];
    int size;
    int length;
};

void Display(struct Array arr)
{
    int i;
    printf("\nElements are\n");
    for(i=0;i<arr.length;i++)
        printf("%d ",arr.A[i]);
}

struct Array* Merge(struct Array *arr1,struct Array *arr2)
{
    int i,j,k;
    i=j=k=0;

    struct Array *arr3=(struct Array *)malloc(sizeof(struct
Array));

    while(i<arr1->length && j<arr2->length)
    {
            if(arr1->A[i]<arr2->A[j])
                arr3->A[k++]=arr1->A[i++];
            else
                arr3->A[k++]=arr2->A[j++];
    }
    for(;i<arr1->length;i++)
        arr3->A[k++]=arr1->A[i];
    for(;j<arr2->length;j++)
        arr3->A[k++]=arr2->A[j];
    arr3->length=arr1->length+arr2->length;
    arr3->size=10;

    return arr3;
}

int main()
{
    struct Array arr1={{2,9,21,28,35},10,5};
    struct Array arr1={{2,3,16,18,28},10,5};
    struct Array *arr3;
```

```
arr3=Merge(&arr1,&arr2);
Display(*arr3);

return 0;
}
```

Set Operations on Arrays

```
struct Array
{
    int A[10];
    int size;
    int length;
};

void Display(struct Array arr)
{
    int i;
    printf("\nElements are\n");
    for(i=0;i<arr.length;i++)
        printf("%d ",arr.A[i]);
}

struct Array* Union(struct Array *arr1,struct Array *arr2)
{
    int i,j,k;
    i=j=k=0;


struct Array *arr3=(struct Array *)malloc(sizeof(struct
Array));

    while(i<arr1->length && j<arr2->length)
    {
        if(arr1->A[i]<arr2->A[j])
            arr3->A[k++]=arr1->A[i++];
        else if(arr2->A[j]<arr1->A[i])
            arr3->A[k++]=arr2->A[j++];
        else
        {
            arr3->A[k++]=arr1->A[i++];
            j++;
        }
    }
    for(;i<arr1->length;i++)
        arr3->A[k++]=arr1->A[i];
    for(;j<arr2->length;j++)
        arr3->A[k++]=arr2->A[j];

    arr3->length=k;
    arr3->size=10;

    return arr3;
```

```
}

struct Array* Intersection(struct Array *arr1,struct Array
*arr2)
{
    int i,j,k;
    i=j=k=0;

    struct Array *arr3=(struct Array *)malloc(sizeof(struct
Array));

    while(i<arr1->length && j<arr2->length)
    {
        if(arr1->A[i]<arr2->A[j])
            i++;
        else if(arr2->A[j]<arr1->A[i])
            j++;
        else if(arr1->A[i]==arr2->A[j])
        {
            arr3->A[k++]=arr1->A[i++];
            j++;
        }
    }

    arr3->length=k;
    arr3->size=10;

    return arr3;
}

struct Array* Difference(struct Array *arr1,struct Array
*arr2)
{
    int i,j,k;
    i=j=k=0;

    struct Array *arr3=(struct Array *)malloc(sizeof(struct
Array));

    while(i<arr1->length && j<arr2->length)
    {
        if(arr1->A[i]<arr2->A[j])
            arr3->A[k++]=arr1->A[i++];
        else if(arr2->A[j]<arr1->A[i])
            j++;
        else
        {
            i++;
```

```
            j++;
        }
    }
    for(;i<arr1->length;i++)
        arr3->A[k++]=arr1->A[i];


    arr3->length=k;
    arr3->size=10;

    return arr3;
}

int main()
{
    struct Array arr1={{2,9,21,28,35},10,5};
    struct Array arr1={{2,3,9,18,28},10,5};
    struct Array *arr3;

arr3=Union(&arr1,&arr2);
Display(*arr3);

return 0;
}
```

Array Menu using C

```c
#include <stdio.h>
#include<stdlib.h>

struct Array
{
    int *A;
    int size;
    int length;
};

void Display(struct Array arr)
{
    int i;
    printf("\nElements are\n");
    for(i=0;i<arr.length;i++)
        printf("%d ",arr.A[i]);
}

void Append(struct Array *arr,int x)
{
    if(arr->length<arr->size)
        arr->A[arr->length++]=x;

}

void Insert(struct Array *arr,int index,int x)
{
    int i;
    if(index>=0 && index <=arr->length)
    {
        for(i=arr->length;i>index;i--)
            arr->A[i]=arr->A[i-1];
        arr->A[index]=x;
        arr->length++;

    }
}
```

```
int Delete(struct Array *arr,int index)
{
    int x=0;
    int i;

    if(index>=0 && index<arr->length)
    {
        x=arr->A[index];
        for(i=index;i<arr->length-1;i++)
            arr->A[i]=arr->A[i+1];
        arr->length--;
        return x;
    }

    return 0;
}

void swap(int *x,int *y)
{
    int temp;
    temp=*x;
    *x=*y;
    *y=temp;
}

int LinearSearch(struct Array *arr,int key)
{
    int i;
    for(i=0;i<arr->length;i++)
    {
        if(key==arr->A[i])
        {
            swap(&arr->A[i],&arr->A[0]);
            return i;
        }
    }
    return -1;
}
```

```
int BinarySearch(struct Array arr,int key)
{
    int l,mid,h;
    l=0;
    h=arr.length-1;

    while(l<=h)
    {
        mid=(l+h)/2;
        if(key==arr.A[mid])
            return mid;
        else if(key<arr.A[mid])
            h=mid-1;
        else
            l=mid+1;
    }
    return -1;
}

int RBinSearch(int a[],int l,int h,int key)
{
    int mid;

    if(l<=h)
    {
        mid=(l+h)/2;
        if(key==a[mid])
            return mid;
        else if(key<a[mid])
            return RBinSearch(a,l,mid-1,key);
        else
            return RBinSearch(a,mid+1,h,key);
    }
    return -1;
}

int Get(struct Array arr,int index)
{
        if(index>=0 && index<arr.length)
            return arr.A[index];
```

```
    return -1;
}

void Set(struct Array *arr,int index,int x)
{
    if(index>=0 && index<arr->length)
        arr->A[index]=x;
}

int Max(struct Array arr)
{
    int max=arr.A[0];
    int i;
    for(i=1;i<arr.length;i++)
    {
        if(arr.A[i]>max)
            max=arr.A[i];
    }
    return max;
}
int Min(struct Array arr)
{
    int min=arr.A[0];
    int i;
    for(i=1;i<arr.length;i++)
    {
        if(arr.A[i]<min)
            min=arr.A[i];
    }
    return min;
}

int Sum(struct Array arr)
{
    int s=0;
    int i;
    for(i=0;i<arr.length;i++)
        s+=arr.A[i];

    return s;
```

```
}

float Avg(struct Array arr)
{
    return (float)Sum(arr)/arr.length;
}

void Reverse(struct Array *arr)
{
    int *B;
    int i,j;

    B=(int *)malloc(arr->length*sizeof(int));
    for(i=arr->length-1,j=0;i>=0;i--,j++)
        B[j]=arr->A[i];
    for(i=0;i<arr->length;i++)
        arr->A[i]=B[i];

}

void Reverse2(struct Array *arr)
{
    int i,j;
    for(i=0,j=arr->length-1;i<j;i++,j--)
    {
        swap(&arr->A[i],&arr->A[j]);
    }
}

void InsertSort(struct Array *arr,int x)
{
    int i=arr->length-1;
    if(arr->length==arr->size)
        return;
    while(i>=0 && arr->A[i]>x)
    {
        arr->A[i+1]=arr->A[i];
        i--;
    }
    arr->A[i+1]=x;
```

```
    arr->length++;

}

int isSorted(struct Array arr)
{
    int i;
    for(i=0;i<arr.length-1;i++)
    {
        if(arr.A[i]>arr.A[i+1])
            return 0;
    }
    return 1;
}

void Rearrange(struct Array *arr)
{
    int i,j;
    i=0;
    j=arr->length-1;

    while(i<j)
    {
        while(arr->A[i]<0)i++;
        while(arr->A[j]>=0)j--;
        if(i<j)swap(&arr->A[i],&arr->A[j]);
    }

}

struct Array* Merge(struct Array *arr1,struct Array
*arr2)
{
    int i,j,k;
    i=j=k=0;

    struct Array *arr3=(struct Array
*)malloc(sizeof(struct Array));

    while(i<arr1->length && j<arr2->length)
```

```
    {
            if(arr1->A[i]<arr2->A[j])
                arr3->A[k++]=arr1->A[i++];
            else
                arr3->A[k++]=arr2->A[j++];
    }
    for(;i<arr1->length;i++)
        arr3->A[k++]=arr1->A[i];
    for(;j<arr2->length;j++)
        arr3->A[k++]=arr2->A[j];
    arr3->length=arr1->length+arr2->length;
    arr3->size=10;

    return arr3;
}


struct Array* Union(struct Array *arr1,struct Array
*arr2)
{
    int i,j,k;
    i=j=k=0;

    struct Array *arr3=(struct Array
*)malloc(sizeof(struct Array));

    while(i<arr1->length && j<arr2->length)
    {
        if(arr1->A[i]<arr2->A[j])
            arr3->A[k++]=arr1->A[i++];
        else if(arr2->A[j]<arr1->A[i])
            arr3->A[k++]=arr2->A[j++];
        else
        {
            arr3->A[k++]=arr1->A[i++];
            j++;
        }
    }
    for(;i<arr1->length;i++)
        arr3->A[k++]=arr1->A[i];
```

```
    for(;j<arr2->length;j++)
        arr3->A[k++]=arr2->A[j];

    arr3->length=k;
    arr3->size=10;

    return arr3;
}

struct Array* Intersection(struct Array *arr1,struct
Array *arr2)
{
    int i,j,k;
    i=j=k=0;

    struct Array *arr3=(struct Array
*)malloc(sizeof(struct Array));

    while(i<arr1->length && j<arr2->length)
    {
        if(arr1->A[i]<arr2->A[j])
            i++;
        else if(arr2->A[j]<arr1->A[i])
            j++;
        else if(arr1->A[i]==arr2->A[j])
        {
            arr3->A[k++]=arr1->A[i++];
            j++;
        }
    }

    arr3->length=k;
    arr3->size=10;

    return arr3;
}

struct Array* Difference(struct Array *arr1,struct
Array *arr2)
{
```

```
    int i,j,k;
    i=j=k=0;

    struct Array *arr3=(struct Array
*)malloc(sizeof(struct Array));

    while(i<arr1->length && j<arr2->length)
    {
        if(arr1->A[i]<arr2->A[j])
            arr3->A[k++]=arr1->A[i++];
        else if(arr2->A[j]<arr1->A[i])
            j++;
        else
        {
            i++;
            j++;
        }
    }
    for(;i<arr1->length;i++)
        arr3->A[k++]=arr1->A[i];


    arr3->length=k;
    arr3->size=10;

    return arr3;
}



int main()
{
    struct Array arr1;
    int ch;
    int x,index;

    printf("Enter Size of Array");
    scanf("%d",&arr1.size);
    arr1.A=(int *)malloc(arr1.size*sizeof(int));
    arr1.length=0;
```

```c
    do
    {
    printf("\n\nMenu\n");
    printf("1. Insert\n");
    printf("2. Delete\n");
    printf("3. Search\n");
    printf("4. Sum\n");
    printf("5. Display\n");
    printf("6.Exit\n");

    printf("enter you choice ");
    scanf("%d",&ch);

    switch(ch)
    {
        case 1: printf("Enter an element and index
");
            scanf("%d%d",&x,&index);
            Insert(&arr1,index,x);
            break;
        case 2: printf("Enter index ");
            scanf("%d",&index);
            x=Delete(&arr1,index);
            printf("Deleted Element is %d\n",x);
            break;
        case 3:printf("Enter element to search ");
            scanf("%d",&x);
            index=LinearSearch(&arr1,x);
            printf("Element index %d",index);
            break;
        case 4:printf("Sum is %d\n",Sum(arr1));
            break;
        case 5:Display(arr1);

    }
    }while(ch<6);
    return 0;
}
```

Array C++ class

```
#include <iostream>
using namespace std;
template<class T>
class Array
{
private:
    T *A;
    int size;
    int length;
public:
    Array()
    {
        size=10;
        A=new T[10];
        length=0;
    }
    Array(int sz)
    {
        size=sz;
        length=0;
        A=new T[size];
    }
    ~Array()
    {
        delete []A;
    }
    void Display();
    void Insert(int index,T x);
    T Delete(int index);
};
template<class T>
void Array<T>::Display()
{
    for(int i=0;i<length;i++)
        cout<<A[i]<<" ";
    cout<<endl;
}
template<class T>
void Array<T>::Insert(int index,T x)
{
    if(index>=0 && index<=length)
    {
        for(int i=length-1;i>=index;i--)
```

```
            A[i+1]=A[i];
        A[index]=x;
        length++;
    }
}
template<class T>
T Array<T>::Delete(int index)
{
    T x=0;
    if(index>=0 && index<length)
    {
        x=A[index];
        for(int i=index;i<length-1;i++)
            A[i]=A[i+1];
        length--;
    }
    return x;
}
int main()
{
    Array<char> arr(10);

    arr.Insert(0,'a');
    arr.Insert(1,'c');
    arr.Insert(2,'d');
    arr.Display();
    cout<<arr.Delete(0)<<endl;
    arr.Display();
    return 0;
}
```

Array using C++ modified

```
#include <iostream>

using namespace std;
class Array
{
private:
    int *A;
    int size;
    int length;
    void swap(int *x,int *y);

public:
    Array()
    {
        size=10;
        length=0;
        A=new int[size];
    }
    Array(int sz)
    {
        size=sz;
        length=0;
        A=new int[size];
    }
    ~Array()
    {
        delete []A;
    }
    void Display();
    void Append(int x);
    void Insert(int index,int x);
    int Delete(int index);
    int LinearSearch(int key);
    int BinarySearch(int key);
   int Get(int index);
    void Set(int index,int x);
```

```
    int Max();
    int Min();
    int Sum();
    float Avg();
    void Reverse();
    void Reverse2();
    void InsertSort(int x);
    int isSorted();
    void Rearrange();
    Array* Merge(Array arr2);
    Array* Union(Array arr2);
    Array* Diff(Array arr2);
    Array* Inter(Array arr2);
};

void Array::Display()
{
    int i;
    cout<<"\nElements are\n";
    for(i=0;i<length;i++)
        cout<<A[i]<<" ";
}

void Array::Append(int x)
{
    if(length<size)
        A[length++]=x;

}

void Array::Insert(int index,int x)
{
    int i;
    if(index>=0 && index <=length)
    {
        for(i=length;i>index;i--)
            A[i]=A[i-1];
        A[index]=x;
        length++;
```

```
    }
}

int Array::Delete(int index)
{
    int x=0;
    int i;

    if(index>=0 && index<length)
    {
        x=A[index];
        for(i=index;i<length-1;i++)
            A[i]=A[i+1];
        length--;
        return x;
    }

    return 0;
}

void Array::swap(int *x,int *y)
{
    int temp;
    temp=*x;
    *x=*y;
    *y=temp;
}

int Array::LinearSearch(int key)
{
    int i;
    for(i=0;i<length;i++)
    {
        if(key==A[i])
        {
            swap(&A[i],&A[0]);
            return i;
        }
    }
    return -1;
```

```
}
int Array::BinarySearch(int key)
{
    int l,mid,h;
    l=0;
    h=length-1;

    while(l<=h)
    {
        mid=(l+h)/2;
        if(key==A[mid])
            return mid;
        else if(key<A[mid])
            h=mid-1;
        else
            l=mid+1;
    }
    return -1;
}

int Array::Get(int index)
{
    if(index>=0 && index<length)
        return A[index];
    return -1;
}
void Array::Set(int index,int x)
{
    if(index>=0 && index< length)
        A[index]=x;
}
int Array::Max()
{
    int max=A[0];
    int i;
    for(i=1;i<length;i++)
    {
```

```
        if(A[i]>max)
            max=A[i];
    }
    return max;
}
int Array::Min()
{
    int min=A[0];
    int i;
    for(i=1;i<length;i++)
    {
        if(A[i]<min)
            min=A[i];
    }
    return min;
}

int Array::Sum()
{
    int s=0;
    int i;
    for(i=0;i<length;i++)
        s+=A[i];

    return s;
}

float Array::Avg()
{
    return (float)Sum()/length;
}

void Array::Reverse()
{
    int *B;
    int i,j;

    B=(int *)malloc(length*sizeof(int));
    for(i=length-1,j=0;i>=0;i--,j++)
        B[j]=A[i];
```

```
    for(i=0;i<length;i++)
        A[i]=B[i];

}
void Array::Reverse2()
{
    int i,j;
    for(i=0,j= length-1;i<j;i++,j--)
    {
        swap(& A[i],& A[j]);
    }
}
void Array::InsertSort(int x)
{
    int i= length-1;
    if( length== size)
        return;
    while(i>=0 &&  A[i]>x)
    {
        A[i+1]= A[i];
        i--;
    }
    A[i+1]=x;
    length++;

}
int Array::isSorted()
{
    int i;
    for(i=0;i<length-1;i++)
    {
        if(A[i]>A[i+1])
            return 0;
    }
    return 1;
}
```

```
void Array::Rearrange()
{
    int i,j;
    i=0;
    j= length-1;

    while(i<j)
    {
        while( A[i]<0)i++;
        while( A[j]>=0)j--;
        if(i<j)swap(& A[i],& A[j]);
    }

}

Array* Array::Merge(Array arr2)
{
    int i,j,k;
    i=j=k=0;

    Array *arr3=new Array(length+arr2.length);

    while(i<length && j<arr2.length)
    {
        if(A[i]<arr2.A[j])
            arr3->A[k++]=A[i++];
        else
            arr3->A[k++]=arr2.A[j++];
    }
    for(;i<length;i++)
        arr3->A[k++]=A[i];
    for(;j<arr2.length;j++)
        arr3->A[k++]=arr2.A[j];
    arr3->length=length+arr2.length;

    return arr3;
}

Array* Array::Union(Array arr2)
```

```
{
    int i,j,k;
    i=j=k=0;

    Array *arr3=new Array(length+arr2.length);

    while(i<length && j<arr2.length)
    {
        if(A[i]<arr2.A[j])
            arr3->A[k++]=A[i++];
        else if(arr2.A[j]<A[i])
            arr3->A[k++]=arr2.A[j++];
        else
        {
            arr3->A[k++]=A[i++];
            j++;
        }
    }
    for(;i<length;i++)
        arr3->A[k++]=A[i];
    for(;j<arr2.length;j++)
        arr3->A[k++]=arr2.A[j];

    arr3->length=k;

    return arr3;
}
Array* Array::Inter(Array arr2)
{
    int i,j,k;
    i=j=k=0;

    Array *arr3=new Array(length+arr2.length);

    while(i<length && j<arr2.length)
    {
        if(A[i]<arr2.A[j])
```

```
            i++;
        else if(arr2.A[j]<A[i])
            j++;
        else if(A[i]==arr2.A[j])
        {
            arr3->A[k++]=A[i++];
            j++;
        }
    }

    arr3->length=k;

    return arr3;
}
Array* Array::Diff(Array arr2)
{
    int i,j,k;
    i=j=k=0;

    Array *arr3=new Array(length+arr2.length);

    while(i<length && j<arr2.length)
    {
        if(A[i]<arr2.A[j])
            arr3->A[k++]=A[i++];
        else if(arr2.A[j]<A[i])
            j++;
        else
        {
            i++;
            j++;
        }
    }
    for(;i<length;i++)
        arr3->A[k++]=A[i];
```

```
    arr3->length=k;

    return arr3;
}

int main()
{
    Array *arr1;
    int ch,sz;
    int x,index;

    cout<<"Enter Size of Array";
    scanf("%d",&sz);
    arr1=new Array(sz);

    do
    {
        cout<<"\n\nMenu\n";
        cout<<"1. Insert\n";
        cout<<"2. Delete\n";
        cout<<"3. Search\n";
        cout<<"4. Sum\n";
        cout<<"5. Display\n";
        cout<<"6.Exit\n";

        cout<<"enter you choice ";
        cin>>ch;

        switch(ch)
        {
            case 1: cout<<"Enter an element and
index ";
                cin>>x>>index;
                arr1->Insert(index,x);
                break;
            case 2: cout<<"Enter index ";
                cin>>index;
                x=arr1->Delete(index);
```

```
                cout<<"Deleted Element is"<<x;
                break;
            case 3:cout<<"Enter element to search
";
                cin>>x;
                index=arr1->LinearSearch(x);
                cout<<"Element index "<<index;
                break;
            case 4:cout<<"Sum is "<<arr1->Sum();
                break;
            case 5:arr1->Display();

        }
    }while(ch<6);
    return 0;
}
```