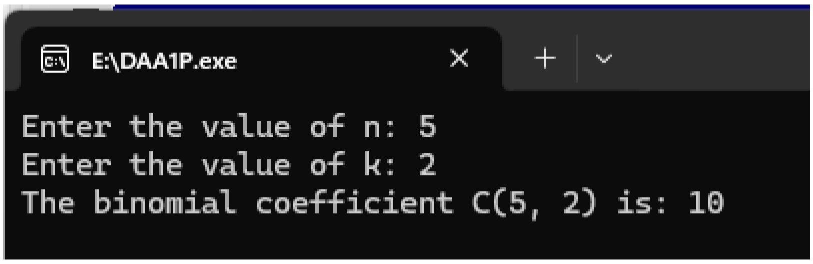
1) Binomial Coefficient (Recursive)

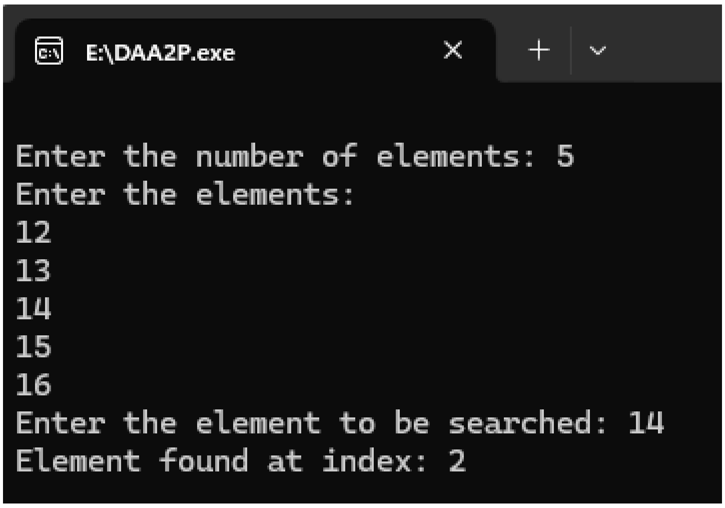
```
#include <iostream>
int binomialCoefficient(int n, int k) {
   // Base cases
   if (k == 0 || k == n) {
      return 1;
   }
   if (k > n) {
      return 0; // Invalid case: k cannot be greater than n
   }
 // Recursive step
   return binomialCoefficient(n - 1, k - 1) + binomialCoefficient(n - 1, k);
}
int main() {
   int n, k;
   std::cout << "Enter the value of n: ";
   std::cin >> n;
   std::cout << "Enter the value of k: ";
   std::cin >> k;
   if (n < 0 || k < 0) {
      std::cout << "n and k must be non-negative." << std::endl;
   } else {
      int result = binomialCoefficient(n, k);
      std::cout << "The binomial coefficient C(" << n << ", " << k << ") is: " << result << std::endl;
   }

   return 0;
}
////////////////output///////////
```



```
Enter the value of n: 5
Enter the value of k: 2
The binomial coefficient C(5, 2) is: 10
```

2) Linear Search(search element)

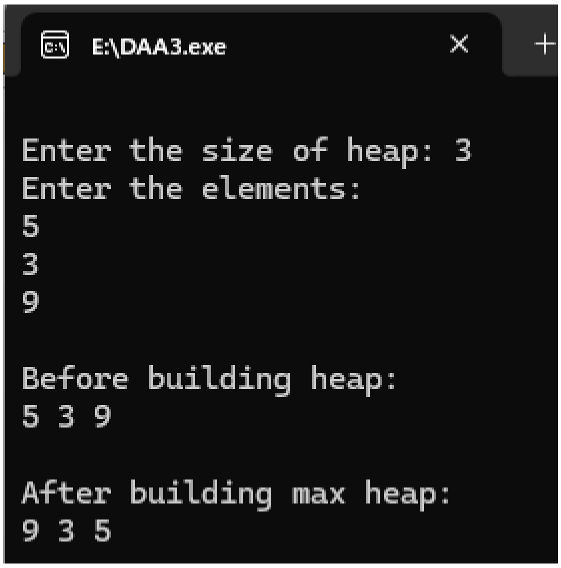
```
#include <iostream>
using namespace std;
class SearchEle {
   int A[50], n, no;
public:
   void Get();
   void Search();
};
void SearchEle::Get() {
   cout << "\nEnter the number of elements: ";
   cin >> n;
   cout << "Enter the elements:\n";
   for (int i = 0; i < n; i++) {
      cin >> A[i];
   }
   cout << "Enter the element to be searched: ";
   cin >> no;
}
void SearchEle::Search() {
   bool found = false;
   for (int i = 0; i < n; i++) {
      if (A[i] == no) {
         cout << "Element found at index: " << i << endl;
         found = true;
         break;
      }
   }
   if (!found) {
      cout << "Element not found.\n";
   }
}
int main() {
   SearchEle s;
   s.Get();
   s.Search();
   return 0;
}
//////////////////////output/////////////////
```



```
Enter the number of elements: 5
Enter the elements:
12
13
14
15
16
Enter the element to be searched: 14
Element found at index: 2
```

3. Max Heap using Insert

```
#include <iostream>
using namespace std;
class InsertMaxHeap {
    int a[20];
    int n;
public:
    void Insert(int);
    void Get();
    void Show();
};
void InsertMaxHeap::Get() {
    cout << "\nEnter the size of heap: ";
    cin >> n;
    cout << "Enter the elements:\n";
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
    }
    cout << "\nBefore building heap:\n";
    Show();

    // Build Max Heap using Insert
    for (int i = 2; i <= n; i++) {
        Insert(i);
    }
    cout << "\nAfter building max heap:\n";
    Show();
}
void InsertMaxHeap::Insert(int i) {
    int temp = a[i];
    int j = i / 2;
    while (j >= 1 && a[j] < temp) {
        a[i] = a[j];
        i = j;
        j = j / 2;
    }
    a[i] = temp;
}
void InsertMaxHeap::Show() {
    for (int i = 1; i <= n; i++) {
        cout << a[i] << " ";
    }
    cout << endl;
}
int main() {
    InsertMaxHeap h;
    h.Get();
    return 0;
}
/////////////////////output/////////////
```

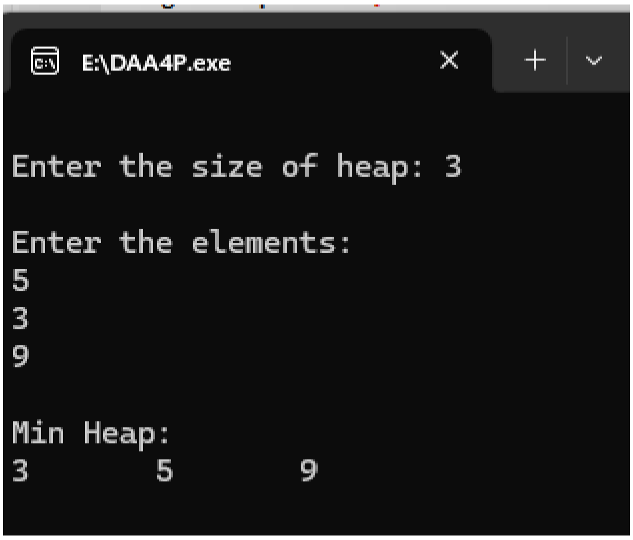


```
Enter the size of heap: 3
Enter the elements:
5
3
9

Before building heap:
5 3 9

After building max heap:
9 3 5
```

4)Elementary Data Structure – Min Heap using Adjust/Heapify

```
#include <iostream>
using namespace std;
class AdjustMinHeap {
   int a[10];
   int n;
public:
   void Adjust(int, int);
   void Heapify(int);
   void Get();
   void Show();
};
void AdjustMinHeap::Get() {
   cout << "\nEnter the size of heap: ";
   cin >> n;

   cout << "\nEnter the elements:\n";
   for (int i = 1; i <= n; i++) {
      cin >> a[i];
   }
   Heapify(n);
   Show();
}
void AdjustMinHeap::Adjust(int i, int n) {
   int j = 2 * i;
   int item = a[i];
   while (j <= n) {
      if (j < n && a[j] > a[j + 1]) {
         j++;
      }
      if (item <= a[j]) {
         break;
      }
      a[i] = a[j];
      i = j;
      j = 2 * i;
   }
   a[i] = item;
}
void AdjustMinHeap::Heapify(int n) {
   for (int i = n / 2; i >= 1; i--) {
      Adjust(i, n);
   }
}
```

```
void AdjustMinHeap::Show() {
   cout << "\nMin Heap:\n";
   for (int i = 1; i <= n; i++) {
      cout << a[i] << "\t";
   }
   cout << endl;
}
int main() {
   AdjustMinHeap a;
   a.Get();
   a.Show();
   return 0;
}
////////////////////////output///////////////////////
```

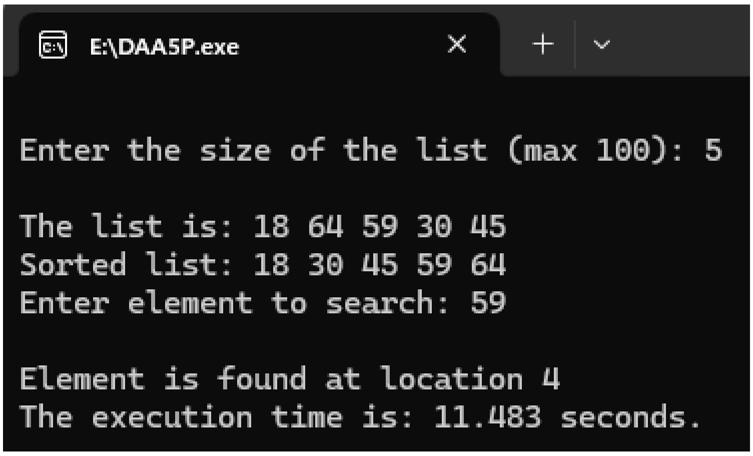


```

Enter the size of heap: 3

Enter the elements:
5
3
9

Min Heap:
3       5       9
```

5)Divide and Conquer – Binary Search

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
class BSearch {
   int A[100], Size;
public:
   void Get();
   void Sort();
   int Search(int, int, int);
   void Show(int);
};
// ?? Get array input and sort it
void BSearch::Get() {
   cout << "\nEnter the size of the list (max 100): ";
   cin >> Size;
   cout << "\nThe list is: ";
   for(int i = 0; i < Size; i++) {
      A[i] = rand() % 100;  // Random numbers between 0 and 99
      cout << A[i] << " ";
   }
   Sort();  // Sort the array
}

// ?? Bubble Sort implementation
void BSearch::Sort() {
   for(int i = 0; i < Size - 1; i++) {
      for(int j = 0; j < Size - i - 1; j++) {
         if(A[j] > A[j + 1]) {
            int temp = A[j];
            A[j] = A[j + 1];
            A[j + 1] = temp;
         }
      }
   }
   cout << "\nSorted list: ";
   for(int i = 0; i < Size; i++)
      cout << A[i] << " ";
}
// ?? Recursive Binary Search
int BSearch::Search(int i, int j, int x) {
   if(i > j)
      return -1;
   int Mid = (i + j) / 2;
```

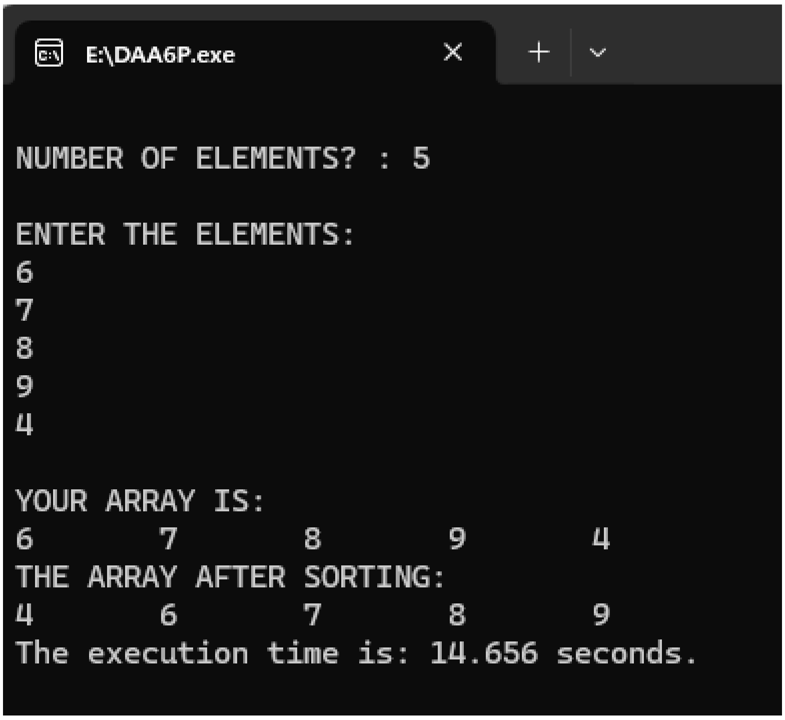
```
    if(x == A[Mid])
        return Mid + 1;  // +1 for human-readable position
    else if(x < A[Mid])
        return Search(i, Mid - 1, x);
    else
        return Search(Mid + 1, j, x);
}
// ?? Show search result
void BSearch::Show(int x) {
    int t = Search(0, Size - 1, x);
    if(t == -1)
        cout << "\nElement is Not Found.";
    else
        cout << "\nElement is found at location " << t;
}
// ?? Main function with execution time
int main() {
    clock_t start, end;
    BSearch b;
    int No;
    srand(time(0));  // Seed for random number generation
    start = clock();  // Start timing
    b.Get();  // Get and sort array
    cout << "\nEnter element to search: ";
    cin >> No;
    b.Show(No);  // Search and display result
    end = clock();  // End timing
    double time_taken = double(end - start) / CLOCKS_PER_SEC;
    cout << "\nThe execution time is: " << time_taken << " seconds.\n";

    return 0;
}
////////////output///////////
```



```
Enter the size of the list (max 100): 5

The list is: 18 64 59 30 45
Sorted list: 18 30 45 59 64
Enter element to search: 59

Element is found at location 4
The execution time is: 11.483 seconds.
```

6.Divide and Conquer – Merge Sort

```
#include <iostream>
#include <ctime>
#include <cmath>
using namespace std;
class Number {
    int a[50], n;
public:
    void getData();
    void mergeSort(int low, int high);
    void merge(int low, int mid, int high);
};

// ?? Input and display function
void Number::getData() {
    cout << "\nNUMBER OF ELEMENTS? : ";
    cin >> n;
    cout << "\nENTER THE ELEMENTS:\n";
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }
    cout << "\nYOUR ARRAY IS:\n";
    for (int i = 0; i < n; i++)
        cout << a[i] << "\t";
    mergeSort(0, n - 1);  // Call merge sort
    cout << "\nTHE ARRAY AFTER SORTING:\n";
    for (int i = 0; i < n; i++)
        cout << a[i] << "\t";
}
// ?? Recursive Merge Sort
void Number::mergeSort(int low, int high) {
    if (low < high) {
        int mid = (low + high) / 2;
        mergeSort(low, mid);
        mergeSort(mid + 1, high);
        merge(low, mid, high);
    }
}
// ?? Merge two sorted halves
void Number::merge(int low, int mid, int high) {
    int h = low, i = low, j = mid + 1;
    int b[50];  // Temporary array
    while (h <= mid && j <= high) {
        if (a[h] <= a[j]) {
            b[i] = a[h];
            h++;
        } else {
            b[i] = a[j];
            j++;
        }
        i++;
    }
    if (h > mid) {
        for (int k = j; k <= high; k++) {
            b[i] = a[k];
            i++;
        }
    } else {
        for (int k = h; k <= mid; k++) {
            b[i] = a[k];
            i++;
        }
    }

    for (int k = low; k <= high; k++) {
        a[k] = b[k];
    }
}
// ?? Main function with execution time
int main() {
    Number obj;
    clock_t start, end;
    start = clock();      // Start timing
    obj.getData();        // Input, sort, and display
    end = clock();        // End timing
    double time_taken = double(end - start) / CLOCKS_PER_SEC;
    cout << "\nThe execution time is: " << time_taken << " seconds.\n";
    return 0;
}
////////////////////output///////////////
```



```
NUMBER OF ELEMENTS? : 5

ENTER THE ELEMENTS:
6
7
8
9
4

YOUR ARRAY IS:
6       7       8       9       4
THE ARRAY AFTER SORTING:
4       6       7       8       9
The execution time is: 14.656 seconds.
```