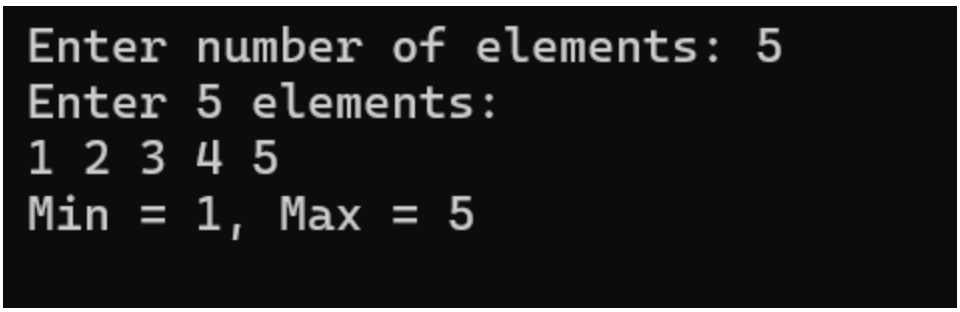
Program-1

```
#include <stdio.h>
int main() {
    int n, a[100];
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter %d elements:\n", n);
    for(int i = 0; i < n; i++)
        scanf("%d", &a[i]);
    int min = a[0], max = a[0];
    for(int i = 1; i < n; i++) {
        if(a[i] < min) min = a[i];
        if(a[i] > max) max = a[i];
    }
    printf("Min = %d, Max = %d\n", min, max);
    return 0;
}
```

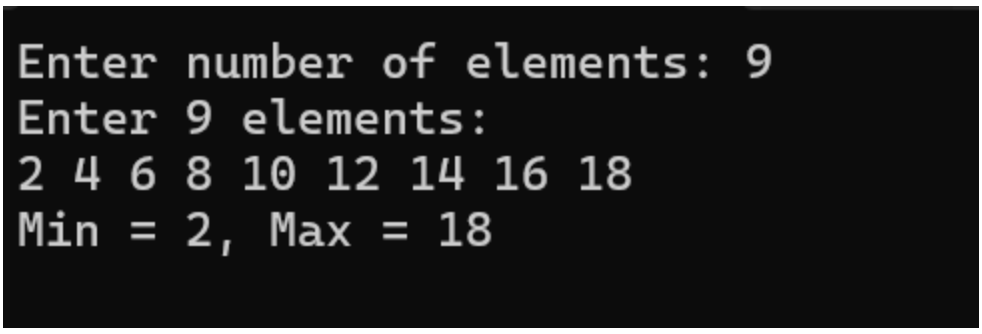
Output:



Program-2:

```
#include <stdio.h>
int main() {
    int n, a[100];
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter %d elements:\n", n);
    for(int i = 0; i < n; i++)
        scanf("%d", &a[i]);
    int min = a[0], max = a[0];
    for(int i = 1; i < n; i++) {
        if(a[i] < min) min = a[i];
        if(a[i] > max) max = a[i];
    }
    printf("Min = %d, Max = %d\n", min, max);
    return 0;
}
```

Output:



Program -3:

```
#include <stdio.h>
void merge(int a[], int l, int m, int r) {
    int i = l, j = m + 1, k = 0;
    int temp[100];
    while(i <= m && j <= r) {
        if(a[i] <= a[j])
            temp[k++] = a[i++];
        else
            temp[k++] = a[j++];
    }
    while(i <= m) temp[k++] = a[i++];
    while(j <= r) temp[k++] = a[j++];
    for(i = l, k = 0; i <= r; i++, k++)
        a[i] = temp[k];
}
void mergeSort(int a[], int l, int r) {
    if(l < r) {
        int m = (l + r) / 2;
        mergeSort(a, l, m);
        mergeSort(a, m + 1, r);
        merge(a, l, m, r);
    }
}
int main() {
    int n, a[100];
```

```
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter %d elements:\n", n);
    for(int i = 0; i < n; i++)
        scanf("%d", &a[i]);
    mergeSort(a, 0, n - 1);
    printf("Sorted Array: ");
    for(int i = 0; i < n; i++)
        printf("%d ", a[i]);
    return 0;
}
```

Output:

```
Enter number of elements: 8
Enter 8 elements:
31 23 35 27 11 21 15 28
Sorted Array: 11 15 21 23 27 28 31 35
```

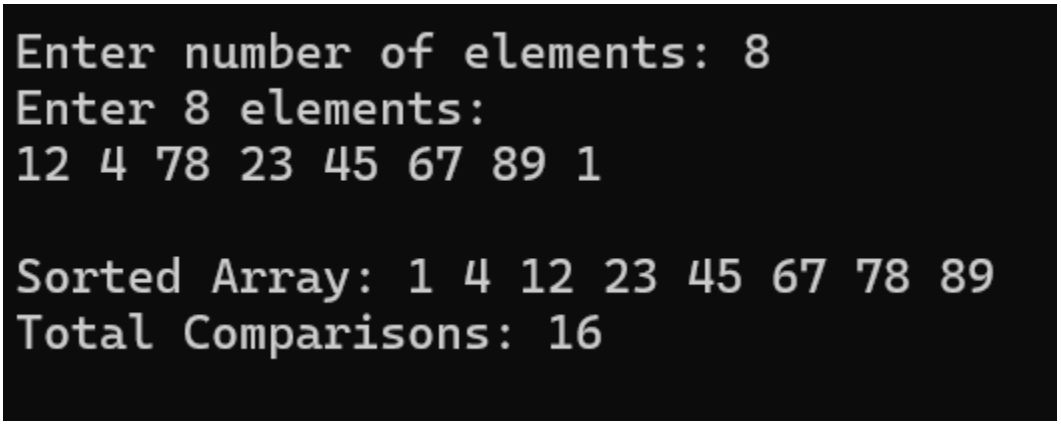
Program -4:

```
#include <stdio.h>
long long comparisons = 0;
void merge(int a[], int l, int m, int r) {
    int i = l, j = m + 1, k = 0;
    int temp[100];
```

```
    while (i <= m && j <= r) {
        comparisons++;
        if (a[i] <= a[j])
            temp[k++] = a[i++];
        else
            temp[k++] = a[j++];
    }
    while (i <= m)
        temp[k++] = a[i++];
    while (j <= r)
        temp[k++] = a[j++];
    for (i = l, k = 0; i <= r; i++, k++)
        a[i] = temp[k];
}
void mergeSort(int a[], int l, int r) {
    if (l < r) {
        int m = (l + r) / 2;
        mergeSort(a, l, m);
        mergeSort(a, m + 1, r);
        merge(a, l, m, r);
    }
}
int main() {
    int n, a[100];
    printf("Enter number of elements: ");
    scanf("%d", &n);
```

```
    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++)
        scanf("%d", &a[i]);
    mergeSort(a, 0, n - 1);
    printf("\nSorted Array: ");
    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\nTotal Comparisons: %lld\n", comparisons);
    return 0;
}
```

Output:



Program-5:

```
#include <stdio.h>
void printArray(int a[], int n) {
    for(int i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\n");
}
int partition(int a[], int low, int high, int n) {
    int pivot = a[low];
    int i = low + 1;
    int j = high;
    int temp;
    while (1) {
        while (i <= high && a[i] <= pivot) i++;
        while (a[j] > pivot) j--;
        if (i >= j) break;
        temp = a[i];
        a[i] = a[j];
        a[j] = temp;
    }
    temp = a[low];
    a[low] = a[j];
    a[j] = temp;
    printf("After partition with pivot %d: ", pivot);
    printArray(a, n);
```

```
    return j;
}
void quickSort(int a[], int low, int high, int n) {
    if (low < high) {
        int p = partition(a, low, high, n);
        quickSort(a, low, p - 1, n);
        quickSort(a, p + 1, high, n);
    }
}
int main() {
    int n, a[100];
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter %d elements:\n", n);
    for(int i = 0; i < n; i++)
        scanf("%d", &a[i]);
    printf("\nQuick Sort steps:\n");
    quickSort(a, 0, n - 1, n);
    printf("\nFinal Sorted Array: ");
    printArray(a, n);
    return 0;
}
```

Output:

```
Enter number of elements: 9
Enter 9 elements:
10 16 8 12 15 6 3 9 5

Quick Sort steps:
After partition with pivot 10: 6 5 8 9 3 10 15 12 16
After partition with pivot 6: 3 5 6 9 8 10 15 12 16
After partition with pivot 3: 3 5 6 9 8 10 15 12 16
After partition with pivot 9: 3 5 6 8 9 10 15 12 16
After partition with pivot 15: 3 5 6 8 9 10 12 15 16

Final Sorted Array: 3 5 6 8 9 10 12 15 16
```

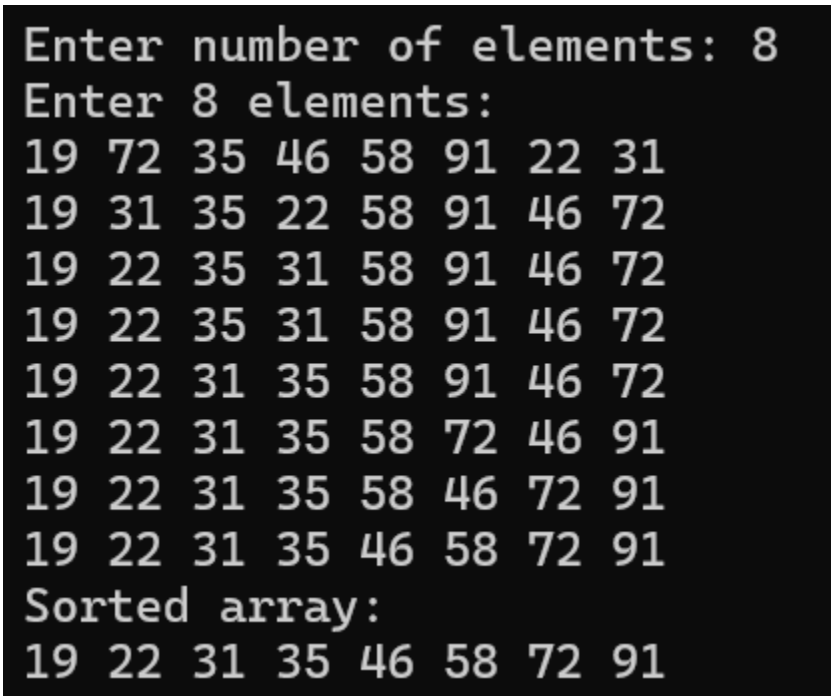
Program -6:

```
#include <stdio.h>
void print(int a[], int n) {
    for(int i=0;i<n;i++) printf("%d ", a[i]);
    printf("\n");
}
int partition(int a[], int low, int high) {
    int mid = (low + high) / 2;
    int pivot = a[mid];
    int i = low, j = high, temp;
    while(i <= j) {
        while(a[i] < pivot) i++;
        while(a[j] > pivot) j--;
```

```
        if(i <= j) {
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
            i++; j--;
        }
    }
    return i;
}
void quickSort(int a[], int low, int high, int n) {
    if(low < high) {
        int index = partition(a, low, high);
        print(a, n);
        quickSort(a, low, index - 1, n);
        quickSort(a, index, high, n);
    }
}
int main() {
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    int a[n];
    printf("Enter %d elements:\n", n);
    for(int i=0;i<n;i++) scanf("%d", &a[i]);
    quickSort(a, 0, n - 1, n);
    printf("Sorted array:\n");
    print(a, n);
```

```
    return 0;
}
```

Output:



Program-7

```
#include <stdio.h>
int main() {
    int n, key, mid, low = 0, high, comp = 0;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    int a[n];
    printf("Enter %d sorted elements:\n", n);
```

```
    for(int i = 0; i < n; i++) scanf("%d", &a[i]);
    printf("Enter search key: ");
    scanf("%d", &key);
    high = n - 1;
    while(low <= high) {
        comp++;
        mid = (low + high) / 2;
        if(a[mid] == key) {
            printf("Element found at position: %d\n", mid + 1);  // 1-based index
            printf("Comparisons: %d\n", comp);
            return 0;
        }
        else if(a[mid] < key)
            low = mid + 1;
        else
            high = mid - 1;
    }
    printf("Element not found\n");
    printf("Comparisons: %d\n", comp);
    return 0;
}
```

Output:

```
Enter number of elements: 9
Enter 9 sorted elements:
a = {5,10,15,20,25,30,35,40,45}
Enter search key: Element found at position: 2
Comparisons: 2
```

Program-8:

```
#include <stdio.h>
int main() {
    int n, key;
    int a[50];
    int low, high, mid;
    int comparisons = 0;
    printf("Enter size of array: ");
    scanf("%d", &n);
    printf("Enter %d sorted elements:\n", n);
    for(int i = 0; i < n; i++)
        scanf("%d", &a[i]);
    printf("Enter search key: ");
    scanf("%d", &key);
    low = 0;
    high = n - 1;
```

```
    while (low <= high) {
        mid = (low + high) / 2;
        comparisons++;
        printf("low = %d, high = %d, mid = %d, a[mid] = %d\n",
                low, high, mid, a[mid]);
        if (a[mid] == key) {
            printf("\nElement found at position %d\n", mid + 1);
            printf("Total comparisons = %d\n", comparisons);
            return 0;
        }
        else if (a[mid] < key) {
            low = mid + 1;
        }
        else {
            high = mid - 1;
        }
    }
    printf("\nElement not found\n");
    printf("Total comparisons = %d\n", comparisons);
    return 0;
}
```

Output:

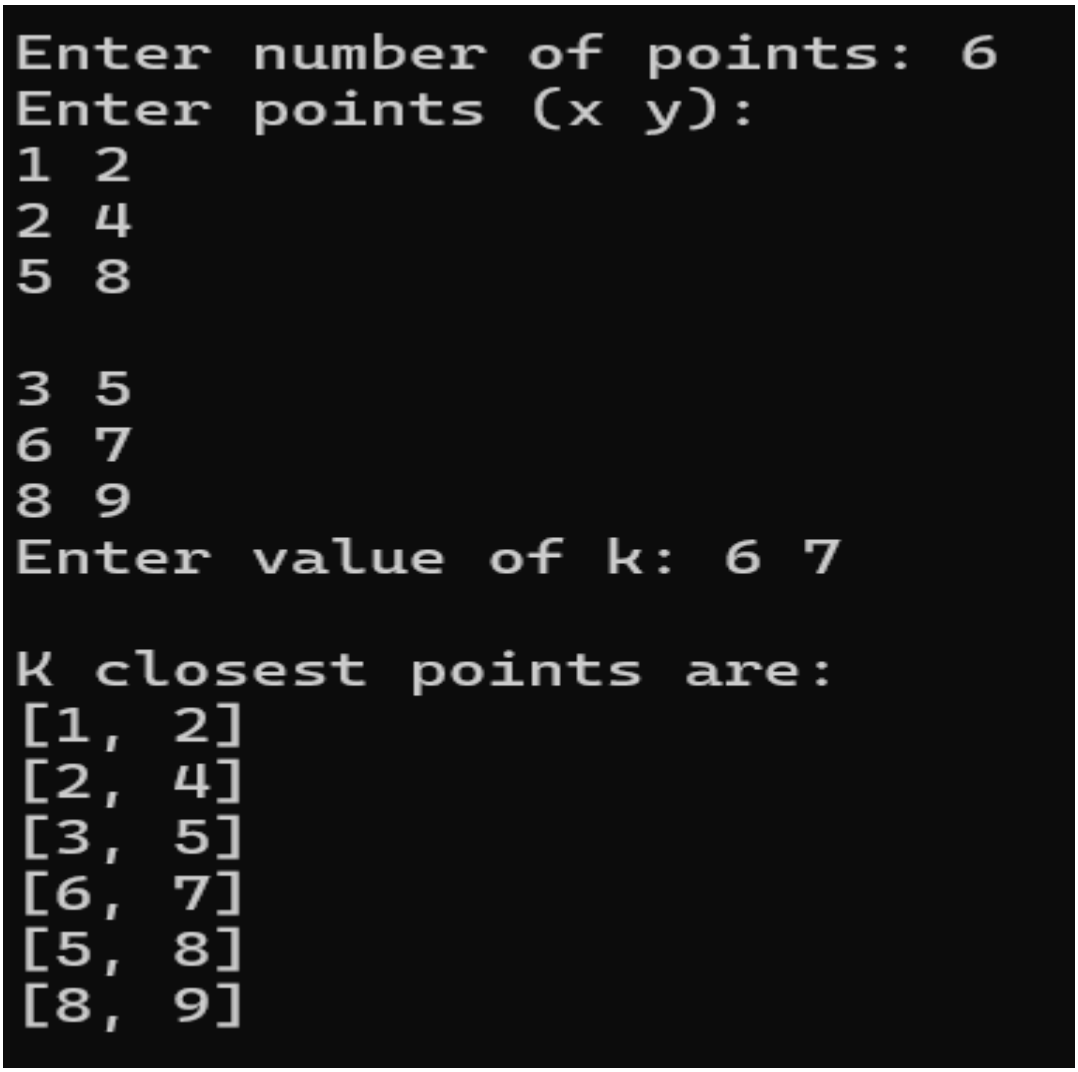
```
Enter size of array: 9
Enter 9 sorted elements:
3 9 14 19 25 31 42 47 53
Enter search key: 42
low = 0, high = 8, mid = 4, a[mid] = 25
low = 5, high = 8, mid = 6, a[mid] = 42

Element found at position 7
Total comparisons = 2
```

Program -9

```
#include <stdio.h>
#include <stdlib.h>
struct Point {
    int x, y;
};
int distance(struct Point p) {
    return p.x * p.x + p.y * p.y;
}
int compare(const void *a, const void *b) {
    struct Point *p1 = (struct Point *)a;
    struct Point *p2 = (struct Point *)b;
    return distance(*p1) - distance(*p2);
```

```
}
int main() {
    int n, k;
    printf("Enter number of points: ");
    scanf("%d", &n);
    struct Point points[n];
    printf("Enter points (x y): \n");
    for (int i = 0; i < n; i++) {
        scanf("%d %d", &points[i].x, &points[i].y);
    }
    printf("Enter value of k: ");
    scanf("%d", &k);
    qsort(points, n, sizeof(struct Point), compare);
    printf("\nK closest points are:\n");
    for (int i = 0; i < k; i++) {
        printf("[%d, %d]\n", points[i].x, points[i].y);
    }
    return 0;
}
```

Output:

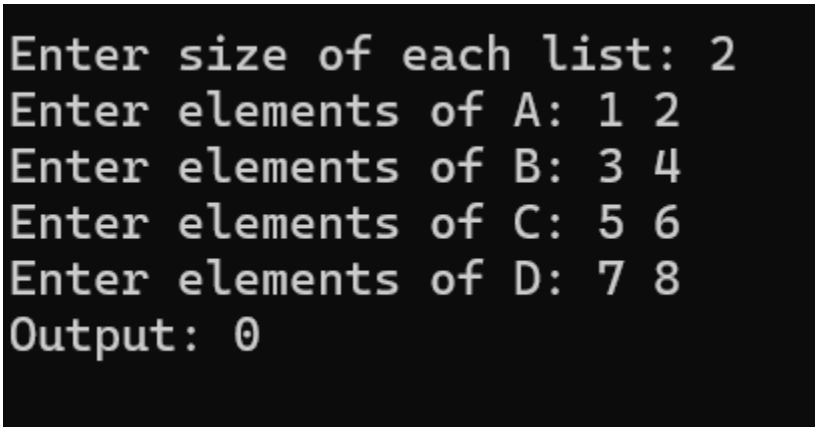

Enter number of points: 6
Enter points (x y):
1 2
2 4
5 8
3 5
6 7
8 9
Enter value of k: 6 7
K closest points are:
[1, 2]
[2, 4]
[3, 5]
[6, 7]
[5, 8]
[8, 9]

Program -10:

```
#include <stdio.h>
int main() {
    int n;
    printf("Enter size of each list: ");
    scanf("%d", &n);
    int A[100], B[100], C[100], D[100];
    printf("Enter elements of A: ");
    for (int i = 0; i < n; i++) scanf("%d", &A[i]);
    printf("Enter elements of B: ");
    for (int i = 0; i < n; i++) scanf("%d", &B[i]);
    printf("Enter elements of C: ");
    for (int i = 0; i < n; i++) scanf("%d", &C[i]);
    printf("Enter elements of D: ");
    for (int i = 0; i < n; i++) scanf("%d", &D[i]);
    int count = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            for (int k = 0; k < n; k++) {
                for (int l = 0; l < n; l++) {
                    if (A[i] + B[j] + C[k] + D[l] == 0) {
                        count++;
                    }
                }
            }
        }
```

```
    }
    printf("Output: %d\n", count);
    return 0;
}
```

Output:



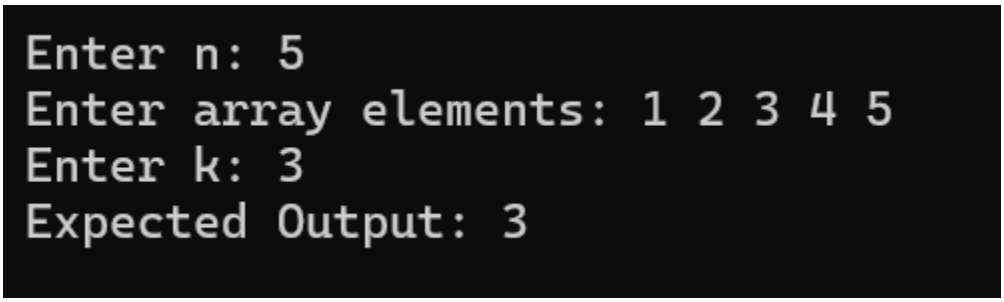
Program-11:

```
#include <stdio.h>
void swap(int *a, int *b) {
    int t = *a; *a = *b; *b = t;
}
void sort5(int arr[], int n) {
    for (int i = 0; i < n - 1; i++)
        for (int j = i + 1; j < n; j++)
            if (arr[i] > arr[j])
                swap(&arr[i], &arr[j]);
```

```
}
int partition(int arr[], int l, int r, int pivot) {
    int i;
    for (i = l; i <= r; i++)
        if (arr[i] == pivot) break;
    swap(&arr[i], &arr[r]);
    i = l;
    for (int j = l; j < r; j++) {
        if (arr[j] < pivot) {
            swap(&arr[i], &arr[j]);
            i++;
        }
    }
    swap(&arr[i], &arr[r]);
    return i;
}
int select_kth(int arr[], int l, int r, int k) {
    if (l == r) return arr[l];
    int n = r - l + 1;
    int medians[100];
    int m = 0;
    for (int i = 0; i < n; i += 5) {
        int group_size = (i + 5 <= n) ? 5 : (n - i);
        sort5(arr + l + i, group_size);
        medians[m++] = arr[l + i + group_size / 2];
    }
```

```
    int median_of_medians =
        (m == 1) ? medians[0] : select_kth(medians, 0, m - 1, m / 2);
    int pos = partition(arr, l, r, median_of_medians);
    int rank = pos - l + 1;
    if (rank == k)
        return arr[pos];
    else if (k < rank)
        return select_kth(arr, l, pos - 1, k);
    else
        return select_kth(arr, pos + 1, r, k - rank);
}
int main() {
    int n, k;
    printf("Enter n: ");
    scanf("%d", &n);
    int arr[200];
    printf("Enter array elements: ");
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);
    printf("Enter k: ");
    scanf("%d", &k);
    int ans = select_kth(arr, 0, n - 1, k);
    printf("Expected Output: %d\n", ans);
    return 0;
}
```

Output:



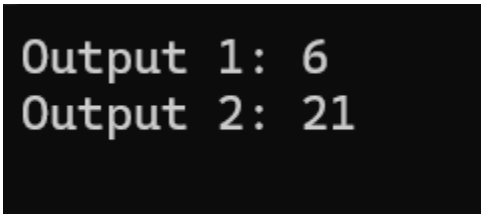
Program-12:

```
#include <stdio.h>
void swap(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}
void sort5(int arr[], int n) {
    for (int i = 0; i < n - 1; i++)
        for (int j = i + 1; j < n; j++)
            if (arr[i] > arr[j])
                swap(&arr[i], &arr[j]);
}
int partition(int arr[], int left, int right, int pivot) {
    int i;
    for (i = left; i <= right; i++)
        if (arr[i] == pivot)
```

```
            break;
    swap(&arr[i], &arr[right]);
    int store = left;
    for (i = left; i < right; i++) {
        if (arr[i] < pivot) {
            swap(&arr[i], &arr[store]);
            store++;
        }
    }
    swap(&arr[store], &arr[right]);
    return store;
}
int select_kth(int arr[], int left, int right, int k) {
    if (left == right)
        return arr[left];
    int n = right - left + 1;
    int medians[100];
    int m = 0;
    for (int i = 0; i < n; i += 5) {
        int size = (i + 5 <= n) ? 5 : (n - i);
        sort5(arr + left + i, size);
        medians[m++] = arr[left + i + size / 2];
    }
    int pivot = (m == 1) ? medians[0] : select_kth(medians, 0, m - 1, m / 2);
    int pos = partition(arr, left, right, pivot);
    int rank = pos - left + 1;
```

```
    if (rank == k)
        return arr[pos];
    else if (k < rank)
        return select_kth(arr, left, pos - 1, k);
    else
        return select_kth(arr, pos + 1, right, k - rank);
}
int median_of_medians(int arr[], int n, int k) {
    return select_kth(arr, 0, n - 1, k);
}
int main() {
    int arr1[] = {1,2,3,4,5,6,7,8,9,10};
    int k1 = 6;
    int n1 = sizeof(arr1)/sizeof(arr1[0]);
    printf("Output 1: %d\n", median_of_medians(arr1, n1, k1));
    int arr2[] = {23,17,31,44,55,21,20,18,19,27};
    int k2 = 5;
    int n2 = sizeof(arr2)/sizeof(arr2[0]);
    printf("Output 2: %d\n", median_of_medians(arr2, n2, k2));
    return 0;
}
```

Output:



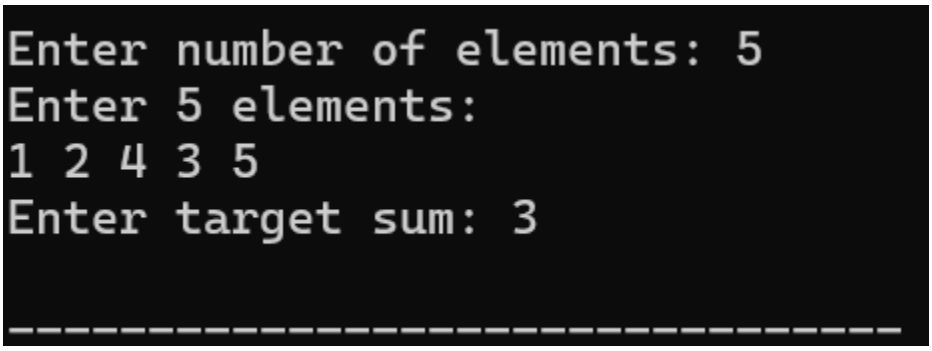
Program-13

```
#include <stdio.h>
#include <stdlib.h>
long long absll(long long x) {
    return x < 0 ? -x : x;
}
int cmp(const void *a, const void *b) {
    long long x = *(long long*)a;
    long long y = *(long long*)b;
    return (x > y) - (x < y);
}
void generate(long long arr[], int n, long long out[], int *idx) {
    int limit = 1 << n;
    for (int mask = 0; mask < limit; mask++) {
        long long sum = 0;
        for (int i = 0; i < n; i++)
            if (mask & (1 << i))
                sum += arr[i];
        out[(*idx)++] = sum;
    }
}
long long meet_middle(long long arr[], int n, long long target) {
    int n1 = n / 2;
    int n2 = n - n1;
    long long A[1 << 20], B[1 << 20];
    int sizeA = 0, sizeB = 0;
```

```
    generate(arr, n1, A, &sizeA);
    generate(arr + n1, n2, B, &sizeB);
    qsort(B, sizeB, sizeof(long long), cmp);
    long long best_sum = 0, best_diff = 1e18;
    for (int i = 0; i < sizeA; i++) {
        long long need = target - A[i];
        int l = 0, r = sizeB - 1;
        while (l <= r) {
            int m = (l + r) / 2;
            long long sum = A[i] + B[m];
            long long diff = absll(sum - target);
            if (diff < best_diff) {
                best_diff = diff;
                best_sum = sum;
            }

            if (sum < target)
                l = m + 1;
            else
                r = m - 1;
        }
    }
    return best_sum;
}
int main() {
    int n;
```

```
    long long target;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    long long arr[n];
    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++)
        scanf("%lld", &arr[i]);
    printf("Enter target sum: ");
    scanf("%lld", &target);
    long long result = meet_middle(arr, n, target);
    printf("\nClosest subset sum to %lld = %lld\n", target, result);
    return 0;
}
```

Output:



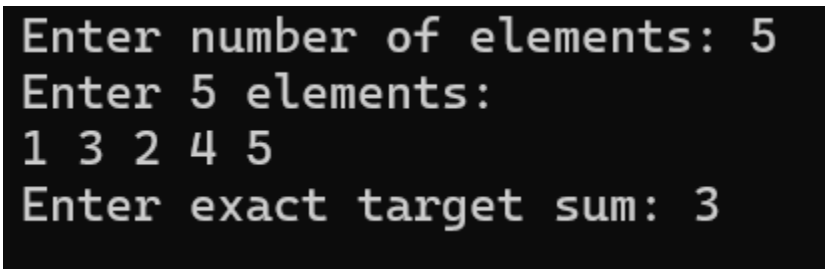
Program-14:

```
#include <stdio.h>
#include <stdlib.h>
int cmp(const void *a, const void *b) {
    long long x = *(long long*)a;
    long long y = *(long long*)b;
    return (x > y) - (x < y);
}
void generate(long long arr[], int n, long long out[], int *idx) {
    int limit = 1 << n;
    for (int mask = 0; mask < limit; mask++) {
        long long sum = 0;
        for (int i = 0; i < n; i++)
            if (mask & (1 << i))
                sum += arr[i];
        out[(*idx)++] = sum;
    }
}
int exists(long long arr[], int size, long long target) {
    int l = 0, r = size - 1;
    while (l <= r) {
        int m = (l + r) / 2;
        if (arr[m] == target) return 1;
        if (arr[m] < target) l = m + 1;
        else r = m - 1;
    }
```

```
    return 0;
}
int meet_middle_exact(long long arr[], int n, long long target) {
    int n1 = n / 2;
    int n2 = n - n1;
    long long A[1 << 20], B[1 << 20];
    int sizeA = 0, sizeB = 0;
    generate(arr, n1, A, &sizeA);
    generate(arr + n1, n2, B, &sizeB);
    qsort(B, sizeB, sizeof(long long), cmp);
    for (int i = 0; i < sizeA; i++) {
        long long need = target - A[i];
        if (exists(B, sizeB, need)) return 1;
    }
    return 0;
}
int main() {
    int n;
    long long target;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    long long arr[n];
    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++)
        scanf("%lld", &arr[i]);
    printf("Enter exact target sum: ");
    scanf("%lld", &target);
```

```
    if (meet_middle_exact(arr, n, target))
        printf("TRUE — A subset exists with sum = %lld\n", target);
    else
        printf("FALSE — No subset sums to %lld\n", target);
    return 0;
}
```

Output:



Program-15

```
#include <stdio.h>
int main() {
    int A[2][2], B[2][2], C[2][2];
    int a, b, c, d, e, f, g;
    printf("Enter matrix A (2x2):\n");
    scanf("%d %d %d %d", &A[0][0], &A[0][1], &A[1][0], &A[1][1]);
    printf("Enter matrix B (2x2):\n");
    scanf("%d %d %d %d", &B[0][0], &B[0][1], &B[1][0], &B[1][1]);
```

```
    a = (A[0][0] + A[1][1]) * (B[0][0] + B[1][1]);
    b = (A[1][0] + A[1][1]) * B[0][0];
    c = A[0][0] * (B[0][1] - B[1][1]);
    d = A[1][1] * (B[1][0] - B[0][0]);
    e = (A[0][0] + A[0][1]) * B[1][1];
    f = (A[1][0] - A[0][0]) * (B[0][0] + B[0][1]);
    g = (A[0][1] - A[1][1]) * (B[1][0] + B[1][1]);
    C[0][0] = a + d - e + g;
    C[0][1] = c + e;
    C[1][0] = b + d;
    C[1][1] = a - b + c + f;
    printf("\nResult Matrix C = A × B:\n");
    printf("%d %d\n", C[0][0], C[0][1]);
    printf("%d %d\n", C[1][0], C[1][1]);
    return 0;
}
```

Output:

```
Enter matrix A (2x2):
1 2
3 4
Enter matrix B (2x2):
5 6
7 8

Result Matrix C = A ├ù B:
19 22
43 50
```

Program-16

```
#include <stdio.h>
#include <math.h>
long karatsuba(long x, long y) {
    if (x < 10 || y < 10)
        return x * y;
    int n = (int)fmax(log10(x) + 1, log10(y) + 1);
    int m = n / 2;
    long p = pow(10, m);
    long a = x / p;
    long b = x % p;
    long c = y / p;
    long d = y % p;
    long ac = karatsuba(a, c);
    long bd = karatsuba(b, d);
    long ad_bc = karatsuba(a + b, c + d) - ac - bd;
    return ac * pow(10, 2 * m) + (ad_bc * p) + bd;
}
int main() {
    long x, y;
    printf("Enter X and Y: ");
    scanf("%ld %ld", &x, &y);
    long result = karatsuba(x, y);
    printf("Product = %ld\n", result);
    return 0;
}
```

Output: