1. Basic Iterative Algorithms: GCD, Fibonacci Sequence, Sequential and Binary Search

```
#include <stdio.h>

int main(void) {
  // GCD
  int a = 36, b = 60;
  while (a != b) {
    a > b ? (a -= b) : (b -= a);
  }
  printf("GCD = %d\n", a);


  // Fibonacci
  int n = 5, t1 = 0, t2 = 1, next;
  printf("Fibonacci: ");
  for (int i = 0; i < n; i++) {
    printf("%d ", t1);
    next = t1 + t2;
    t1 = t2;
    t2 = next;
  }
  printf("\n");


  // Sequential Search
  int arr[] = {1, 4, 7, 8, 9}, key = 7;
  for (int i = 0; i < 5; i++) {
    if (arr[i] == key) {
      printf("Sequential: Found at %d\n", i);
      break; // optional: stop after first matc
```

```c
        }}
    // Binary Search
    int low = 0, high = 4, mid;
    key = 8;
    while (low <= high) {
        mid = (low + high) / 2;
        if (arr[mid] == key) {
            printf("Binary: Found at %d\n", mid);
            break;
        } else if (arr[mid] < key) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }

    return 0;
}
```

Output

GCD = 12

Fibonacci: 0 1 1 2 3

Sequential: Found at 2

Binary: Found at 3

2. Basic Iterative Sorting Algorithms: Bubble Sort, Selection Sort, Insertion Sort

```
#include <stdio.h>

// Function to print array

void printArray(int arr[], int size) {

    for(int i = 0; i < size; i++)

        printf("%d ", arr[i]);

    printf("\n");

}


int main() {

    int bubble[] = {5, 3, 8, 4, 2};

    int selection[] = {5, 3, 8, 4, 2};

    int insertion[] = {5, 3, 8, 4, 2};

    int n = 5;


    // Bubble Sort

    for(int i = 0; i < n-1; i++) {

        for(int j = 0; j < n-i-1; j++) {

            if(bubble[j] > bubble[j+1]) {

                int temp = bubble[j];

                bubble[j] = bubble[j+1];

                bubble[j+1] = temp;

            }

}

 }


    printf("Bubble Sort:    ");

    printArray(bubble, n);
```

```
// Selection Sort
  for(int i = 0; i < n-1; i++) {
     int minIndex = i;
     for(int j = i+1; j < n; j++) {
        if(selection[j] < selection[minIndex])
           minIndex = j;
     }
     int temp = selection[i];
     selection[i] = selection[minIndex];
     selection[minIndex] = temp;
  }
  printf("Selection Sort: ");
  printArray(selection, n);

  // Insertion Sort
  for(int i = 1; i < n; i++) {
     int key = insertion[i];
     int j = i - 1;
     while(j >= 0 && insertion[j] > key) {
        insertion[j + 1] = insertion[j];
        j--;
     }
     insertion[j + 1] = key;
  }
  printf("Insertion Sort: ");
  printArray(insertion, n);
  return 0;
}
```

OUTPUT

Bubble Sort: 2 3 4 5 8

Selection Sort: 2 3 4 5 8

Insertion Sort: 2 3 4 5 8

3. Binary Search with Divide and Conquer Approach

```
#include <stdio.h>

int main() {

  int arr[] = {1, 3, 5, 7, 9}; // Must be sorted

  int n = 5, key = 7, low = 0, high = n - 1, mid;

  printf("Binary Search (Divide and Conquer) for %d: ", key);

  while (low <= high) {

    mid = (low + high) / 2;

    if (arr[mid] == key) {

      printf("Found at index %d\n", mid);

      return 0;

    }

    if (arr[mid] < key)

      low = mid + 1;

    else

      high = mid - 1;

  }

  printf("Not found\n");

  return 0;

}
```

OUTPUT

Binary Search (Divide and Conquer) for 7: Found at index 3

4. Merge Sort, Heap Sort, Quick Sort, Randomized Quick Sort

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void print(int a[], int n) {

   for(int i=0;i<n;i++) printf("%d ", a[i]);

   printf("\n");

}

// Merge Sort

void merge(int a[], int l, int m, int r) {

   int i=l, j=m+1, k=0, temp[10];

   while(i<=m && j<=r) temp[k++] = a[i]<a[j] ? a[i++] : a[j++];

   while(i<=m) temp[k++] = a[i++];

   while(j<=r) temp[k++] = a[j++];

   for(i=l, k=0; i<=r; i++) 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);

   }}

// Heap Sort

void heapify(int a[], int n, int i) {

   int l=2*i+1, r=2*i+2, largest=i;

   if(l<n && a[l]>a[largest]) largest = l;

   if(r<n && a[r]>a[largest]) largest = r;

   if(largest != i) {

int t=a[i]; a[i]=a[largest]; a[largest]=t;

      heapify(a,n,largest);

   }}

void heapSort(int a[], int n) {
```

```
  for(int i=n/2-1;i>=0;i--) heapify(a,n,i);
  for(int i=n-1;i>0;i--) {
    int t=a[0]; a[0]=a[i]; a[i]=t;
    heapify(a,i,0);
  }}
// Quick Sort
int partition(int a[], int l, int h) {
  int p=a[h], i=l-1;
  for(int j=l;j<h;j++) if(a[j]<=p) {
    int t=a[++i]; a[i]=a[j]; a[j]=t;
  }
  int t=a[i+1]; a[i+1]=a[h]; a[h]=t;
  return i+1; }
void quickSort(int a[], int l, int h) {
  if(l<h) {
    int p = partition(a,l,h);
    quickSort(a,l,p-1);
    quickSort(a,p+1,h);
  }}
// Randomized Quick Sort
int randPartition(int a[], int l, int h) {
  int r = l + rand() % (h-l+1);
  int t=a[r]; a[r]=a[h]; a[h]=t;
  return partition(a,l,h);
}
void randQuickSort(int a[], int l, int h) {
  if(l<h) {
    int p = randPartition(a,l,h);
    randQuickSort(a,l,p-1);
    randQuickSort(a,p+1,h);
  }}
int main() {
  srand(time(0));
```

```
    int a1[] = {4,3,2,1}, a2[] = {8,7,6,5,}, a3[] = {12,11,10,9}, a4[] = {16,15,13,14};
    int n = 4;

    mergeSort(a1,0,n-1);
    printf("Merge Sort: "); print(a1,n);

    heapSort(a2,n);
    printf("Heap Sort:  "); print(a2,n);

    quickSort(a3,0,n-1);
    printf("Quick Sort: "); print(a3,n);

    randQuickSort(a4,0,n-1);
    printf("Rand QSort: "); print(a4,n);

    return 0;
}
```

OUTPUT

```
Merge Sort: 1 2 3 4
Heap Sort:  5 6 7 8
Quick Sort: 9 10 11 12
Rand QSort: 13 14 15 16
```

5. Selection Problem with Divide and Conquer Approach

```
#include <stdio.h>

int partition(int arr[], int low, int high) {
   int pivot = arr[high], i = low - 1, j, temp;
   for (j = low; j < high; j++) {
      if (arr[j] <= pivot) {
         i++;
         temp = arr[i];
         arr[i] = arr[j];
         arr[j] = temp;      }   }
   temp = arr[i + 1];
   arr[i + 1] = arr[high];
   arr[high] = temp;
   return i + 1; }
int quickSelect(int arr[], int low, int high, int k) {
   if (low == high) return arr[low];
   int pi = partition(arr, low, high);
   if (k == pi) return arr[k];
   else if (k < pi) return quickSelect(arr, low, pi - 1, k);
   else return quickSelect(arr, pi + 1, high, k);
}
int main() {
   int arr[] = {5, 2, 9, 1, 7};
   int n = 5, k = 2; // Find 3rd smallest (k is 0-based)
   printf("Selection Problem (k=%d smallest): ", k + 1);
   int result = quickSelect(arr, 0, n - 1, k);
   printf("%d\n", result);
   return 0;  }
```

OUTPUT

Selection Problem (k=3 smallest): 5

6.Fractional Knapsack Problem, Job Sequencing with Deadline, Kruskal's Algorithm, Prim's Algorithm, Dijkstra's Algorithm

```
#include <stdio.h>

#include <limits.h>

#include <string.h>

// ---------- 1. Fractional Knapsack ----------

struct Item {

  int weight, profit;

};

void fractionalKnapsack() {

  struct Item items[] = {{10, 60}, {20, 100}, {30, 120}};

  int n = 3, W = 50;

  float ratio[3], total = 0;

  for(int i=0;i<n;i++)

    ratio[i] = (float)items[i].profit/items[i].weight;

  for(int i=0;i<n-1;i++)

    for(int j=i+1;j<n;j++)

      if(ratio[i]<ratio[j]) {

        struct Item t = items[i]; items[i] = items[j]; items[j] = t;

        float r = ratio[i]; ratio[i] = ratio[j]; ratio[j] = r;

      }

  for(int i=0;i<n;i++) {

    if(W >= items[i].weight) {

      total += items[i].profit;

      W -= items[i].weight;

    } else {

      total += ratio[i] * W;

      break;
```

```
    } }
   printf("\n1. Fractional Knapsack Max Profit = %.2f\n", total);
}
// ---------- 2. Job Sequencing ----------
struct Job {
   char id;
   int deadline, profit;
};
void jobSequencing() {
   struct Job jobs[] = {{'a',2,100},{'b',1,19},{'c',2,27},{'d',1,25},{'e',3,15}};
   int n = 5, slot[10];
   memset(slot, -1, sizeof(slot));
   for(int i=0;i<n-1;i++)
      for(int j=i+1;j<n;j++)
         if(jobs[i].profit < jobs[j].profit) {
            struct Job t = jobs[i]; jobs[i] = jobs[j]; jobs[j] = t;
         }
   printf("\n2. Job Sequence: ");
   for(int i=0;i<n;i++) {
      for(int j=jobs[i].deadline-1;j>=0;j--) {
         if(slot[j]==-1) {
            slot[j] = i;
            printf("%c ", jobs[i].id);
            break;
         }
} }
   printf("\n");
}
```

```
// ---------- 3. Kruskal's Algorithm ----------
int parent[10];
int find(int i) {
   while(i != parent[i]) i = parent[i];
   return i;
}
void kruskal() {
   int n = 4;
   int cost[4][4] = {
      {0, 10, 6, 5},
      {10, 0, 0, 15},
      {6, 0, 0, 4},
      {5, 15, 4, 0}
   };
   for(int i=0;i<n;i++) parent[i] = i;
   int edge = 0, mincost = 0;
   printf("\n3. Kruskal's MST Edges:\n");
   while(edge < n-1) {
      int min = 999, a=-1, b=-1;
      for(int i=0;i<n;i++)
         for(int j=0;j<n;j++)
            if(cost[i][j] && cost[i][j]<min && find(i)!=find(j)) {
               min = cost[i][j];
               a = i; b = j;
            }
      if(a != -1 && b != -1) {
         parent[find(a)] = find(b);
         printf("Edge %d-%d = %d\n", a, b, min);
```

```
    mincost += min;
      edge++;
      cost[a][b] = cost[b][a] = 999;
    }
  }
  printf("Kruskal Min Cost = %d\n", mincost);
}

// ---------- 4. Prim's Algorithm ----------
void prim() {
  int cost[5][5] = {
    {0, 2, 0, 6, 0},
    {2, 0, 3, 8, 5},
    {0, 3, 0, 0, 7},
    {6, 8, 0, 0, 9},
    {0, 5, 7, 9, 0}
  };
  int n = 5, selected[5] = {1,0,0,0,0}, edge = 0, total = 0;

  printf("\n4. Prim's MST Edges:\n");
  while(edge < n-1) {
    int min = INT_MAX, x=-1, y=-1;
    for(int i=0;i<n;i++)
      if(selected[i])
        for(int j=0;j<n;j++)
          if(!selected[j] && cost[i][j] && cost[i][j]<min) {
            min = cost[i][j]; x = i; y = j;
          }
```

```
        selected[y] = 1;
        printf("Edge %d-%d = %d\n", x, y, min);
        total += min;
        edge++;
    }
    printf("Prim Min Cost = %d\n", total);
}

// ---------- 5. Dijkstra's Algorithm ----------
void dijkstra() {
    int n = 5, src = 0;
    int cost[5][5] = {
        {0, 10, 0, 5, 0},
        {0, 0, 1, 2, 0},
        {0, 0, 0, 0, 4},
        {0, 3, 9, 0, 2},
        {7, 0, 6, 0, 0}
    };
    int dist[5], visited[5] = {0};
    for(int i=0;i<n;i++) dist[i] = INT_MAX;
    dist[src] = 0;

    for(int i=0;i<n-1;i++) {
        int u=-1, min=INT_MAX;
        for(int j=0;j<n;j++)
            if(!visited[j] && dist[j]<min) {
                min = dist[j]; u = j;
            }
```

```
        visited[u] = 1;
        for(int v=0;v<n;v++)
            if(cost[u][v] && dist[u]+cost[u][v] < dist[v])
                dist[v] = dist[u] + cost[u][v];
    }

    printf("\n5. Dijkstra (source: 0):\n");
    for(int i=0;i<n;i++)
        printf("To %d = %d\n", i, dist[i]);
}

// ---------- MAIN FUNCTION ----------
int main() {
    fractionalKnapsack();
    jobSequencing();
    kruskal();
    prim();
    dijkstra();
    return 0; }
```

OUTPUT

1. Fractional Knapsack Max Profit = 240.00

2. Job Sequence: a c e

3. Kruskal's MST Edges:

Edge 2-3 = 4

Edge 0-3 = 5

Edge 0-1 = 10

Kruskal Min Cost = 19

4. Prim's MST Edges:

Edge 0-1 = 2

Edge 1-2 = 3

Edge 1-4 = 5

Edge 0-3 = 6

Prim Min Cost = 16

5. Dijkstra (source: 0):

To 0 = 0

To 1 = 8

To 2 = 9

To 3 = 5

To 4 = 7

7. Implement the Dynamic Programming Algorithms

```c
#include <stdio.h>

int main() {
    int n = 10, i;
    int dp[n];
    dp[0] = 0; dp[1] = 1;
    printf("Fibonacci (Dynamic Programming) first %d numbers: ", n);
    for (i = 2; i < n; i++)
        dp[i] = dp[i - 1] + dp[i - 2];
    for (i = 0; i < n; i++)
        printf("%d ", dp[i]);
    printf("\n");
    return 0;
}
```

OUTPUT

Fibonacci (Dynamic Programming) first 10 numbers: 0 1 1 2 3 5 8 13 21 34

8. Algorithms Using Backtracking Approach

```c
#include <stdio.h>

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

void permute(int arr[], int start, int end) {
    int i;
    if (start == end) {
        for (i = 0; i <= end; i++)
            printf("%d ", arr[i]);
        printf("\n");
    } else {
        for (i = start; i <= end; i++) {
            swap(&arr[start], &arr[i]);
            permute(arr, start + 1, end);
            swap(&arr[start], &arr[i]);
        }
    }
}

int main() {
    int arr[] = {1, 2, 3};
    int n = 3;
    printf("Permutations (Backtracking):\n");
    permute(arr, 0, n - 1);
    return 0;
}
```

OUTPUT

```
Permutations (Backtracking):
1 2 3
1 3 2
2 1 3
2 3 1
3 2 1
3 1 2
```

9. Implement Approximation Algorithm

```c
#include <stdio.h>

int main() {

    int edges[3][2] = {{0, 1}, {1, 2}, {2, 0}}; // Graph as edge list

    int n = 3, i, covered[n], vertexCover[n];

    for (i = 0; i < n; i++) {

        covered[i] = 0;

        vertexCover[i] = 0;

    }

    printf("Vertex Cover (Approximation):\n");

    for (i = 0; i < 3; i++) {

        int u = edges[i][0], v = edges[i][1];

        if (!covered[u] || !covered[v]) {

            if (!covered[u]) vertexCover[u] = 1;

            if (!covered[v]) vertexCover[v] = 1;

            covered[u] = 1;

            covered[v] = 1;

        }

    }

    for (i = 0; i < n; i++)

        if (vertexCover[i])

            printf("Vertex %d\n", i);

    return 0;

}
```

OUTPUT

```
Vertex Cover (Approximation):

Vertex 0

Vertex 1

Vertex 2
```