**Radix Sort Algorithm**

Radix\_sort (list, n)

shift = 1

for loop = 1 to keysize do

for entry = 1 to n do

  bucketnumber = (list[entry].key / shift) mod 10

  append (bucket[bucketnumber], list[entry])

list = combinebuckets()

shift = shift \* 10

**Program**

**// Radix Sort in C Programming**

#include<stdio.h>

int get\_max (int a[], int n){

   int max = a[0];

   for (int i = 1; i < n; i++)

      if (a[i] > max)

         max = a[i];

   return max;

}

void radix\_sort (int a[], int n){

   int bucket[10][10], bucket\_cnt[10];

   int i, j, k, r, NOP = 0, divisor = 1, lar, pass;

   lar = get\_max (a, n);

   while (lar > 0){

      NOP++;

      lar /= 10;

   }

   for (pass = 0; pass < NOP; pass++){

      for (i = 0; i < 10; i++){

         bucket\_cnt[i] = 0;

      }

      for (i = 0; i < n; i++){

         r = (a[i] / divisor) % 10;

         bucket[r][bucket\_cnt[r]] = a[i];

         bucket\_cnt[r] += 1;

      }

      i = 0;

      for (k = 0; k < 10; k++){

         for (j = 0; j < bucket\_cnt[k]; j++){

            a[i] = bucket[k][j];

            i++;

         }

      }

      divisor \*= 10;

      printf ("After pass %d : ", pass + 1);

      for (i = 0; i < n; i++)

         printf ("%d ", a[i]);

      printf ("\n");

   }

}

int main (){

   int i, n, a[10];

   printf ("Enter the number of items to be sorted: ");

   scanf ("%d", &n);

   printf ("Enter items: ");

   for (i = 0; i < n; i++){

      scanf ("%d", &a[i]);

   }

   radix\_sort (a, n);

   printf ("Sorted items : ");

   for (i = 0; i < n; i++)

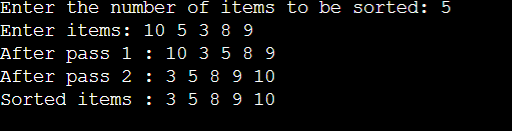
      printf ("%d ", a[i]);

   printf ("\n");

   return 0;

}

**Output:**



|  |  |
| --- | --- |
| **Time Complexity** |  |
| Best | O(n+k) |
| Worst | O(n+k) |
| Average | O(n+k) |
| **Space Complexity** | O(max) |
| **Stability** | Yes |

**Shell sort**

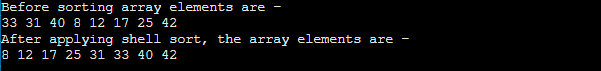
**Algorithm**

1. ShellSort(a, n) // 'a' is the given array, 'n' is the size of array
2. for (interval = n/2; interval > 0; interval /= 2)
3. for ( i = interval; i < n; i += 1)
4. temp = a[i];
5. for (j = i; j >= interval && a[j - interval] > temp; j -= interval)
6. a[j] = a[j - interval];
7. a[j] = temp;
8. End ShellSort

**Code:**

1. #include <stdio.h>
2. /\* function to implement shellSort \*/
3. int shell(int a[], int n)
4. {
5. /\* Rearrange the array elements at n/2, n/4, ..., 1 intervals \*/
6. for (int interval = n/2; interval > 0; interval /= 2)
7. {
8. for (int i = interval; i < n; i += 1)
9. {
10. /\* store a[i] to the variable temp and make the ith position empty \*/
11. int temp = a[i];
12. int j;
13. for (j = i; j >= interval && a[j - interval] > temp; j -= interval)
14. a[j] = a[j - interval];
16. // put temp (the original a[i]) in its correct position
17. a[j] = temp;
18. }
19. }
20. return 0;
21. }
22. void printArr(int a[], int n) /\* function to print the array elements \*/
23. {
24. int i;
25. for (i = 0; i < n; i++)
26. printf("%d ", a[i]);
27. }
28. int main()
29. {
30. int a[] = { 33, 31, 40, 8, 12, 17, 25, 42 };
31. int n = sizeof(a) / sizeof(a[0]);
32. printf("Before sorting array elements are - \n");
33. printArr(a, n);
34. shell(a, n);
35. printf("\nAfter applying shell sort, the array elements are - \n");
36. printArr(a, n);
37. return 0;
38. }

**OUTPUT:**



|  |  |
| --- | --- |
| **Case** | **Time Complexity** |
| **Best Case** | O(n\*logn) |
| **Average Case** | O(n\*log(n)2) |
| **Worst Case** | O(n2) |

# **Bubble sort**

# **Algorithm**

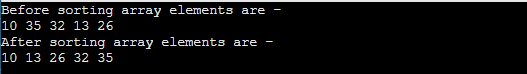
1. Begin BubbleSort(arr)
2. for all array elements
3. if arr[i] > arr[i+1]
4. swap(arr[i], arr[i+1])
5. end if
6. end for
7. return arr
8. end BubbleSort

|  |  |
| --- | --- |
| **Case** | **Time Complexity** |
| **Best Case** | O(n) |
| **Average Case** | O(n2) |
| **Worst Case** | O(n2) |

**Program :**

1. #include<stdio.h>
2. void print(int a[], int n) //function to print array elements
3. {
4. int i;
5. for(i = 0; i < n; i++)
6. {
7. printf("%d ",a[i]);
8. }
9. }
10. void bubble(int a[], int n) // function to implement bubble sort
11. {
12. int i, j, temp;
13. for(i = 0; i < n; i++)
14. {
15. for(j = i+1; j < n; j++)
16. {
17. if(a[j] < a[i])
18. {
19. temp = a[i];
20. a[i] = a[j];
21. a[j] = temp;
22. }
23. }
24. }
25. }
26. void main ()
27. {
28. int i, j,temp;
29. int a[5] = { 10, 35, 32, 13, 26};
30. int n = sizeof(a)/sizeof(a[0]);
31. printf("Before sorting array elements are - \n");
32. print(a, n);
33. bubble(a, n);
34. printf("\nAfter sorting array elements are - \n");
35. print(a, n);
36. }

**OUTPUT:**



# **Matrix multiplication in C**

**Program**

#include <stdio.h>

// function to get matrix elements entered by the user

void getMatrixElements(int matrix[][10], int row, int column) {

printf("\nEnter elements: \n");

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

for (int j = 0; j < column; ++j) {

printf("Enter a%d%d: ", i + 1, j + 1);

scanf("%d", &matrix[i][j]);

}

}

}

// function to multiply two matrices

void multiplyMatrices(int first[][10],

int second[][10],

int result[][10],

int r1, int c1, int r2, int c2) {

// Initializing elements of matrix mult to 0.

for (int i = 0; i < r1; ++i) {

for (int j = 0; j < c2; ++j) {

result[i][j] = 0;

}

}

// Multiplying first and second matrices and storing it in result

for (int i = 0; i < r1; ++i) {

for (int j = 0; j < c2; ++j) {

for (int k = 0; k < c1; ++k) {

result[i][j] += first[i][k] \* second[k][j];}}}}

// function to display the matrix

void display(int result[][10], int row, int column) {

printf("\nOutput Matrix:\n");

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

for (int j = 0; j < column; ++j) {

printf("%d ", result[i][j]);

if (j == column - 1)

printf("\n");}}}

int main() {

int first[10][10], second[10][10], result[10][10], r1, c1, r2, c2;

printf("Enter rows and column for the first matrix: ");

scanf("%d %d", &r1, &c1);

printf("Enter rows and column for the second matrix: ");

scanf("%d %d", &r2, &c2);

// Taking input until

// 1st matrix columns is not equal to 2nd matrix row

while (c1 != r2) {

printf("Error! Enter rows and columns again.\n");

printf("Enter rows and columns for the first matrix: ");

scanf("%d%d", &r1, &c1);

printf("Enter rows and columns for the second matrix: ");

scanf("%d%d", &r2, &c2);

}

// get elements of the first matrix

getMatrixElements(first, r1, c1);

// get elements of the second matrix

getMatrixElements(second, r2, c2);

// multiply two matrices.

multiplyMatrices(first, second, result, r1, c1, r2, c2);

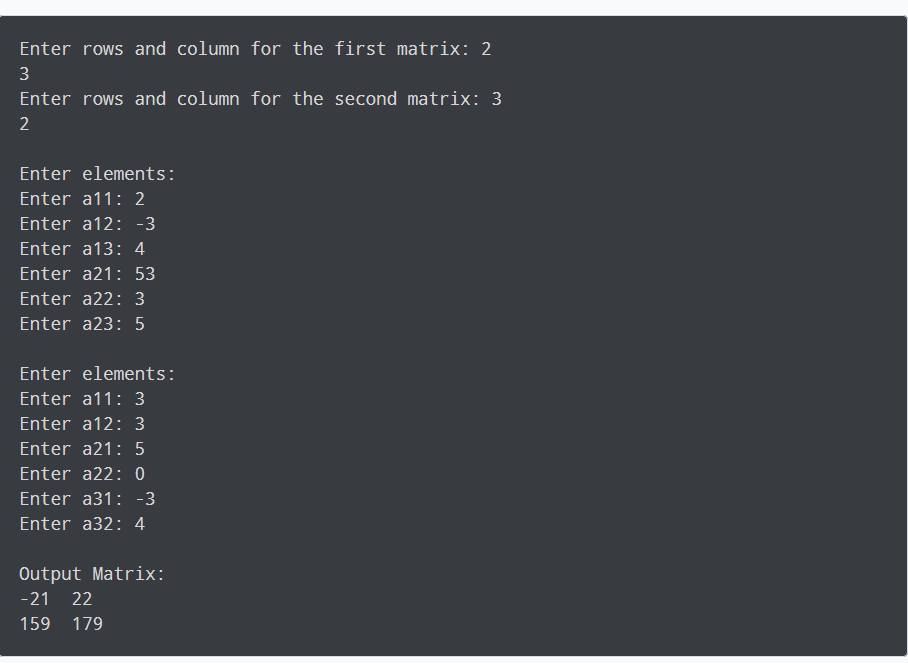
// display the result

display(result, r1, c2);

return 0;

}

**Output:**



## Longest Common Subsequence

## Algorithm:

X and Y be two given sequences

Initialize a table LCS of dimension X.length \* Y.length

X.label = X

Y.label = Y

LCS[0][] = 0

LCS[][0] = 0

Start from LCS[1][1]

Compare X[i] and Y[j]

If X[i] = Y[j]

LCS[i][j] = 1 + LCS[i-1, j-1]

Point an arrow to LCS[i][j]

Else

LCS[i][j] = max(LCS[i-1][j], LCS[i][j-1])

Point an arrow to max(LCS[i-1][j], LCS[i][j-1])

**Program:**

// The longest common subsequence in C

#include <stdio.h>

#include <string.h>

int i, j, m, n, LCS\_table[20][20];

char S1[20] = "ACADB", S2[20] = "CBDA", b[20][20];

void lcsAlgo() {

m = strlen(S1);

n = strlen(S2);

// Filling 0's in the matrix

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

LCS\_table[i][0] = 0;

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

LCS\_table[0][i] = 0;

// Building the mtrix in bottom-up way

for (i = 1; i <= m; i++)

for (j = 1; j <= n; j++) {

if (S1[i - 1] == S2[j - 1]) {

LCS\_table[i][j] = LCS\_table[i - 1][j - 1] + 1;

} else if (LCS\_table[i - 1][j] >= LCS\_table[i][j - 1]) {

LCS\_table[i][j] = LCS\_table[i - 1][j];

} else {

LCS\_table[i][j] = LCS\_table[i][j - 1];

}

}

int index = LCS\_table[m][n];

char lcsAlgo[index + 1];

lcsAlgo[index] = '\0';

int i = m, j = n;

while (i > 0 && j > 0) {

if (S1[i - 1] == S2[j - 1]) {

lcsAlgo[index - 1] = S1[i - 1];

i--;

j--;

index--;

}

else if (LCS\_table[i - 1][j] > LCS\_table[i][j - 1])

i--;

else

j--;

}

// Printing the sub sequences

printf("S1 : %s \nS2 : %s \n", S1, S2);

printf("LCS: %s", lcsAlgo);

}

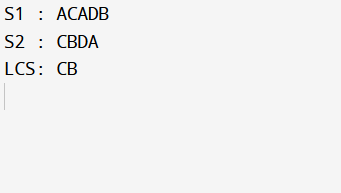
int main() {

lcsAlgo();

printf("\n");

}

**Output:**



**Knapsack:**

**Program:**

#include<stdio.h>

int max(int a, int b) { return (a > b)? a : b; }

int knapSack(int W, int wt[], int val[], int n)

{

   int i, w;

   int K[n+1][W+1];

   for (i = 0; i <= n; i++)

   {

       for (w = 0; w <= W; w++)

       {

           if (i==0 || w==0)

               K[i][w] = 0;

           else if (wt[i-1] <= w)

                 K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]],  K[i-1][w]);

           else

                 K[i][w] = K[i-1][w];

       }

   }

   return K[n][W];

}

int main()

{

    int i, n, val[20], wt[20], W;

    printf("Enter number of items:");

    scanf("%d", &n);

    printf("Enter value and weight of items:\n");

    for(i = 0;i < n; ++i){

     scanf("%d%d", &val[i], &wt[i]);

    }

    printf("Enter size of knapsack:");

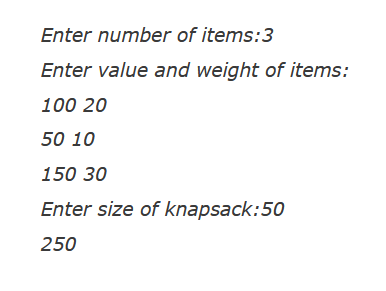
    scanf("%d", &W);

    printf("%d", knapSack(W, wt, val, n));

    return 0;

}

**Output**



# **Selection Sort**

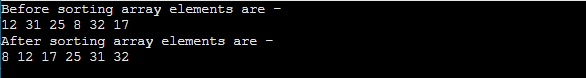
# **Algorithm**

1. SELECTION SORT(arr, n)
3. Step 1: Repeat Steps 2 **and** 3 **for** i = 0 to n-1
4. Step 2: CALL SMALLEST(arr, i, n, pos)
5. Step 3: SWAP arr[i] with arr[pos]
6. [END OF LOOP]
7. Step 4: EXIT
9. SMALLEST (arr, i, n, pos)
10. Step 1: [INITIALIZE] SET SMALL = arr[i]
11. Step 2: [INITIALIZE] SET pos = i
12. Step 3: Repeat **for** j = i+1 to n
13. **if** (SMALL > arr[j])
14. SET SMALL = arr[j]
15. SET pos = j
16. [END OF **if**]
17. [END OF LOOP]
18. Step 4: RETURN pos

**Program:**

1. #include <stdio.h>
3. **void** selection(**int** arr[], **int** n)
4. {
5. **int** i, j, small;
7. **for** (i = 0; i < n-1; i++)    // One by one move boundary of unsorted subarray
8. {
9. small = i; //minimum element in unsorted array
11. **for** (j = i+1; j < n; j++)
12. **if** (arr[j] < arr[small])
13. small = j;
14. // Swap the minimum element with the first element
15. **int** temp = arr[small];
16. arr[small] = arr[i];
17. arr[i] = temp;
18. }
19. }
20. **void** printArr(**int** a[], **int** n) /\* function to print the array \*/
21. {
22. **int** i;
23. **for** (i = 0; i < n; i++)
24. printf("%d ", a[i]);
25. }
27. **int** main()
28. {
29. **int** a[] = { 12, 31, 25, 8, 32, 17 };
30. **int** n = **sizeof**(a) / **sizeof**(a[0]);
31. printf("Before sorting array elements are - \n");
32. printArr(a, n);
33. selection(a, n);
34. printf("\nAfter sorting array elements are - \n");
35. printArr(a, n);
36. **return** 0;
37. }

**OUTPUT:**



|  |  |
| --- | --- |
| **TIME COMPLEXITY:**  **Best Case** | O(n2) |
| **Average Case** | O(n2) |
| **Worst Case** | O(n2) |