Tehnici de sortare

```
void combSort(int a[], int n){
        int gap=n;
        bool swapped=true;
        while(gap !=1 || swapped == true){
                gap=getNextGap(gap);
        swapped=false;
        for(int i=0; i<n-gap; i++){
                if(a[i]>a[i+gap]){
                    swap(a[i], a[i+gap]);
        swapped=true;
        }}}
```

bubble sort

```
int n, v[100];
    //citire v[] cu n elemente
    bool sortat;
    do{
            sortat = true;
    for(int i = 0 ; i < n - 1 ; i ++)
        if(v[i] > v[i+1]){
            int aux = v[i];
            v[i] = v[i+1];
            v[i+1] = aux;
            sortat = false;
            }
        }
        while(!sortat);
```

Program Fără pointeri - Bubble Sort / Comb Sort

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

```
int *alocare_memorie(int dimensiune)
{
    int *tab = (int*)malloc(dimensiune * sizeof(int));
    if(tab == NULL)
    {
        printf("Eroare! Nu s-a putut aloca memorie pentru tablou.\n");
        exit(0);
    }
    return tab;
}
void introducere_tastatura(int *tab, int dimensiune)
{
    printf("Introduceti valorile tabloului:\n");
    for(int i = 0; i < dimensiune; i++)
    {
        scanf("%d", &tab[i]);
```

```
    }
}
void completare_random(int *tab, int dimensiune)
{
    srand(time(NULL));
    for(int i = 0; i < dimensiune; i++)
    {
        tab[i] = rand() % 100;
    }
}
void bubble_sort(int tab[], int dimensiune)
{
    int i, j, temp;
    for(i = 0; i < dimensiune-1; i++)
    {
        for(j = 0; j < dimensiune-i-1; j++)
        {
            if(tab[j] > tab[j+1])
            {
                temp = tab[j];
                tab[j] = tab[j+1];
                tab[j+1] = temp;
            }
        }
    }
}
void comb_sort(int tab[], int dimensiune)
{
    int gap = dimensiune;
    int i, j, temp;
    int swapped = 1;
    while(gap > 1 || swapped)
    {
        gap = gap * 10 / 13;
        if(gap == 9 || gap == 10) gap = 11;
        if(gap < 1) gap = 1;
        swapped = 0;
        for(i = 0, j = gap; j < dimensiune; i++, j++)
        {
            if(tab[i] > tab[j])
            {
                temp = tab[i];
                tab[i] = tab[j];
                tab[j] = temp;
                swapped = 1;
            }
        }
    }
}
void eliberare_memorie(int *tab)
{
    free(tab);
    printf("Memoria a fost eliberata cu succes.\n");
    exit(0);
}
void afisare_meniu(int *tab, int dimensiune)
{
    int optiune;
    printf("\n1. Alocare dinamica a memoriei\n");
```

```
    printf("2. Introducere valorilor tabloului de la tastatura\n");
    printf("3. Completare tablou cu valori random\n");
    printf("4. Sortare prin Bubble Sort\n");
    printf("5. Sortare prin Comb Sort\n");
    printf("6. Eliberare memoria si iesire din program\n");
    printf("7. Afisarea tabloului\n");
printf("Introduceti optiunea: ");
scanf("%d", &optiune);
switch(optiune)
{
    case 1:
        printf("Introduceti dimensiunea tabloului: ");
        scanf("%d", &dimensiune);
        tab = alocare_memorie(dimensiune);
        printf("Memorie alocata cu succes pentru un tablou cu %d elemente.\n",
dimensiune);
        break;
    case 2:
        if(tab == NULL)
        {
            printf("Eroare! Tabloul nu a fost alocat.\n");
        }
        else
        {
            introducere_tastatura(tab, dimensiune);
            printf("Valorile au fost introduse cu succes.\n");
        }
        break;
    case 3:
        if(tab == NULL)
        {
            printf("Eroare! Tabloul nu a fost alocat.\n");
        }
        else
        {
            completare_random(tab, dimensiune);
            printf("Valorile au fost completate cu succes.\n");
        }
        break;
    case 4:
        if(tab == NULL)
        {
            printf("Eroare! Tabloul nu a fost alocat.\n");
        }
        else
        {
            bubble_sort(tab, dimensiune);
            printf("Valorile au fost sortate cu succes prin Bubble Sort.\n");
        }
        break;
    case 5:
        if(tab == NULL)
        {
            printf("Eroare! Tabloul nu a fost alocat.\n");
        }
        else
        {
            comb_sort(tab, dimensiune);
            printf("Valorile au fost sortate cu succes prin Comb Sort.\n");
```

```
        }
        break;
    case 6:
        if(tab == NULL)
        {
            printf("Eroare! Tabloul nu a fost alocat.\n");
        }
        else
        {
            eliberare_memorie(tab);
        }
        break;
    case 7:
    if(tab == NULL)
    {
        printf("Eroare! Tabloul nu a fost alocat.\n");
    }
    else
    {
        printf("Valorile din tablou sunt: ");
        for(int i = 0; i < dimensiune; i++)
        {
            printf("%d ", tab[i]);
        }
        printf("\n");
    }
    break;
    default:
        printf("Optiunea selectata nu exista.\n");
        break;
      }
      afisare_meniu(tab, dimensiune);
}
int main() {
      int *tab = NULL;
      int dimensiune = 0;
      afisare_meniu(tab, dimensiune);
      return 0;
}
```

Program Cu pointeri - Bubble Sort / Comb Sort

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

int *alocare_memorie(int dimensiune)
{
    int *tab = (int*)malloc(dimensiune * sizeof(int));
    if(tab == NULL)
    {
        printf("Eroare! Nu s-a putut aloca memorie pentru tablou.\n");
        exit(0);
    }
    return tab;
}
void introducere_tastatura(int *tab, int dimensiune)
{
    printf("Introduceti valorile tabloului:\n");
```

```
    for(int i = 0; i < dimensiune; i++)
    {
        scanf("%d", &tab[i]);
    }
}
void completare_random(int *tab, int dimensiune)
{
    srand(time(NULL));
    for(int i = 0; i < dimensiune; i++)
    {
        tab[i] = rand() % 100;
    }
}
void bubble_sort(int *tab, int dimensiune)
{
    int i, j, temp;
    for(i = 0; i < dimensiune-1; i++)
    {
        for(j = 0; j < dimensiune-i-1; j++)
        {
            if(*(tab+j) > *(tab+j+1))
            {
                temp = *(tab+j);
                *(tab+j) = *(tab+j+1);
                *(tab+j+1) = temp;
            }
        }
    }
}
void comb_sort(int *tab, int dimensiune)
{
    int gap = dimensiune;
    int i, j, temp;
    int swapped = 1;
    while(gap > 1 || swapped)
    {
        gap = gap * 10 / 13;
        if(gap == 9 || gap == 10) gap = 11;
        if(gap < 1) gap = 1;

        swapped = 0;
        for(i = 0, j = gap; j < dimensiune; i++, j++)
        {
            if(*(tab+i) > *(tab+j))
            {
                temp = *(tab+i);
                *(tab+i) = *(tab+j);
                *(tab+j) = temp;
                swapped = 1;
            }
        }
    }
}
void eliberare_memorie(int *tab)
{
    free(tab);
    printf("Memoria a fost eliberata cu succes.\n");
    exit(0);
}
```

```
void afisare_meniu(int *tab, int dimensiune)
{
    int optiune;
    printf("\n1. Alocare dinamica a memoriei\n");
    printf("2. Introducere valorilor tabloului de la tastatura\n");
    printf("3. Completare tablou cu valori random\n");
    printf("4. Sortare prin Bubble Sort\n");
    printf("5. Sortare prin Comb Sort\n");
    printf("6. Eliberare memoria si iesire din program\n");
    printf("7. Afisarea tabloului\n");
printf("Introduceti optiunea: ");
scanf("%d", &optiune);
switch(optiune)
{
    case 1:
        printf("Introduceti dimensiunea tabloului: ");
        scanf("%d", &dimensiune);
        tab = alocare_memorie(dimensiune);
        printf("Memorie alocata cu succes pentru un tablou cu %d elemente.\n",
dimensiune);
        break;
    case 2:
        if(tab == NULL)
        {
            printf("Eroare! Tabloul nu a fost alocat.\n");
        }
        else
        {
            introducere_tastatura(tab, dimensiune);
            printf("Valorile au fost introduse cu succes.\n");
        }
        break;
    case 3:
        if(tab == NULL)
        {
            printf("Eroare! Tabloul nu a fost alocat.\n");
        }
        else
        {
            completare_random(tab, dimensiune);
            printf("Valorile au fost completate cu succes.\n");
        }
        break;
    case 4:
        if(tab == NULL)
        {
            printf("Eroare! Tabloul nu a fost alocat.\n");
        }
        else
        {
            bubble_sort(tab, dimensiune);
            printf("Valorile au fost sortate cu succes prin Bubble Sort.\n");
        }
        break;
    case 5:
        if(tab == NULL)
        {
            printf("Eroare! Tabloul nu a fost alocat.\n");
        }
```

```
        else
        {
            comb_sort(tab, dimensiune);
            printf("Valorile au fost sortate cu succes prin Comb Sort.\n");
        }
        break;
    case 6:
        if(tab == NULL)
        {
            printf("Eroare! Tabloul nu a fost alocat.\n");
        }
        else
        {
            eliberare_memorie(tab);
        }
        break;
    case 7:
    if(tab == NULL)
    {
        printf("Eroare! Tabloul nu a fost alocat.\n");
    }
    else
    {
        printf("Valorile din tablou sunt: ");
        for(int i = 0; i < dimensiune; i++)
        {
            printf("%d ", tab[i]);
        }
        printf("\n");
    }
    break;
    default:
        printf("Optiunea selectata nu exista.\n");
        break;
      }
      afisare_meniu(tab, dimensiune);
}
int main() {
      int *tab = NULL;
      int dimensiune = 0;
      afisare_meniu(tab, dimensiune);
      return 0;
}
```

Radix Sort / Counting Sort

```
#include <stdio.h>
void read_vector(int v[], int n) {
    printf("Introduceti %d elemente in vector:\n", n);
    for (int i = 0; i < n; i++) {
        printf("v[%d] = ", i);
        scanf("%d", &v[i]);
    }
}

void print_vector(int v[], int n) {
```

```
    printf("Vectorul este: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", v[i]);
    }
    printf("\n");
}
```

```
void radix_sort(int v[], int n) {
    int i, m = v[0], exp = 1;
    int bucket[n], b[n];
    for (i = 0; i < n; i++) {
        if (v[i] > m)
            m = v[i];
    }
    while (m / exp > 0) {
        int bucket_count[10] = {0};
        for (i = 0; i < n; i++)
            bucket_count[v[i] / exp % 10]++;
        for (i = 1; i < 10; i++)
            bucket_count[i] += bucket_count[i - 1];
        for (i = n - 1; i >= 0; i--)
            b[--bucket_count[v[i] / exp % 10]] = v[i];
        for (i = 0; i < n; i++)
            v[i] = b[i];
        exp *= 10;
    }
}
```

```
void counting_sort(int v[], int n) {
    int i, max = v[0];
    for (i = 1; i < n; i++) {
        if (v[i] > max)
            max = v[i];
    }
    int bucket[max + 1];
    for (i = 0; i <= max; i++) {
        bucket[i] = 0;
    }
    for (i = 0; i < n; i++) {
        bucket[v[i]]++;
    }
    int j = 0;
    for (i = max; i >= 0; i--) {
        while (bucket[i] > 0) {
            v[j++] = i;
            bucket[i]--;
        }
```

```
    }
}
```

```
int main() {
    int v[10], n = 10;
    read_vector(v, n);
    print_vector(v, n);

    radix_sort(v, n);
    printf("Vectorul sortat folosind Radix Sort este: ");
    print_vector(v, n);

    counting_sort(v, n);
    printf("Vectorul sortat folosind Counting Sort este: ");
    print_vector(v, n);

    return 0;
}
```

Quick Sort

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

void swap (int *x, int *y);
void quicksort (int array[], int length);
void quicksort_recursion(int array[], int low, int high);
int partition (int array[], int low, int high);
```

```
    int main () {

        int a[] = {10,-3,23,7,8,15,3,9,22,45,56,99,-12};
        int length = 13;

        printf ("Array-ul initial:\n");
        for (int i = 0; i < length; i++)
            printf ("%d ", a[i]);
            printf ("\n\n");

        quicksort (a, length);
        printf ("Array-ul sortat prin Quick Sort:\n");

        for (int i = 0; i < length; i++)
            printf("%d ", a[i]);
            printf("\n");
        return 0;
    }
void swap (int *x, int *y)
{
  int temp = *x;
```

```
  *x = *y;
  *y = temp;
}
void quicksort (int array[], int length)
{
    srand(time(NULL));
    quicksort_recursion(array, 0, length - 1);
}
void quicksort_recursion(int array[], int low, int high)
{
  if (low < high)
  {
    int pivot_index = partition(array, low, high);
    quicksort_recursion(array, low, pivot_index - 1);
    quicksort_recursion(array, pivot_index + 1, high);
  }
}
int partition(int array[], int low, int high)
{
    int pivot_index = low + (rand() % (high - low));
    if (pivot_index != high)
        swap(&array[pivot_index], &array[high]);

    int pivot_value = array[high];
    int i = low;

    for (int j = low; j < high; j++)
    {
        if (array[j] <= pivot_value)
        {
            swap(&array[i], &array[j]);
            i++;
        }
    }
    swap(&array[i], &array[high]);
    return i;
}
```

Merge Sort

```
#include <stdio.h>
#define max 10
int a[10] = { 10, 14, 19, 26, 27, 33, 35, 42, 44, 0 };
int b[10];
void merging(int low, int mid, int high) {
   int d1 = low, d2 = mid + 1, i;
   for(i = low; d1 <= mid && d2 <= high; i++) {
```

```
/* if(a[d1] <= a[d2]) {
         b[i] = a[d1];
         d1++;
        } else {
            b[i] = a[d2];
            d2++;
            }*/
```

```
// o alta versiune pentru if
        if(a[d1] <= a[d2])
                b[i] = a[d1++];
            else
                b[i] = a[d2++];
```

```
   } // End For Loop
   while(d1 <= mid)
      b[i++] = a[d1++];

   while(d2 <= high)
      b[i++] = a[d2++];

   for(i = low; i <= high; i++)
      a[i] = b[i];
} // End function merging
```

```
void sort(int low, int high) {
   int mid;

   if(low < high) {
      mid = (low + high) / 2;
      sort(low, mid);
      sort(mid+1, high);
      merging(low, mid, high);
   }
}
```

```
int main() {
   int i;

   printf("List before sorting\n");
   for(i = 0; i < max; i++)
      printf("%d ", a[i]);

   sort(0, max - 1);

   printf("\nList after sorting\n");
   for(i = 0; i < max; i++)
      printf("%d ", a[i]);
}
```

Selection Sort / Heap Sort

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

// functia de citire a unui vector de lungime n, dat ca pointer
void citire_vector (int *v, int n) {
    printf ("Introduceti elementele vectorului: ");
    for (int i = 0; i < n; i++){
        scanf ("%d", v + i);
    }
}
```

```
// functia de afisare a unui vector de lungime n, dat ca pointer
void afisare_vector (int *v, int n) {
    printf ("Elementele vectorului sunt: ");
```

```
    for (int i = 0; i < n; i++) {
        printf ("%d ", *(v + i));
    }
    printf ("\n");
}
```

```
// functia de sortare a unui vector de lungime n prin selection sort
void selection_sort (int *v, int n) {
    for (int i = 0; i < n - 1; i++) {
        int min = i;
        for (int j = i + 1; j < n; j++)  {
            if (*(v + j) < *(v + min))  {
                min = j;
            }
        }
        if (min != i) {
            int temp = *(v + i);
            *(v + i) = *(v + min);
            *(v + min) = temp;
        }
    }
}
```

```
// functia auxiliara folosita in heap_sort pentru a mentine proprietatea de max-heap
void heapify (int *v, int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;
    if (left < n && *(v + left) > *(v + largest)) {
        largest = left;
    }
    if (right < n && *(v + right) > *(v + largest)) {
        largest = right;
    }
    if (largest != i) {
        int temp = *(v + i);
        *(v + i) = *(v + largest);
        *(v + largest) = temp;
        heapify (v, n, largest);
    }
}
```

```
// functia de sortare a unui vector de lungime n prin heap sort
void heap_sort (int * v, int n) {
    for (int i = n / 2 - 1; i >= 0; i--) {
        heapify (v, n, i);
    }
    for (int i = n - 1; i >= 0; i--) {
        int temp = *(v);
        *(v) = *(v + i);
        *(v + i) = temp;
        heapify (v, i, 0);
    }
}
```

```
// functia de afisare a meniului si de selectare a optiunii
```

```
int meniu () {
    int optiune;
    printf ("\n");
    printf ("1. Introduceti primul vector\n");
    printf ("2. Introduceti al doilea vector\n");
    printf ("3. Afisati primul vector\n");
    printf ("4. Afisati al doilea vector\n");
    printf ("5. Sortati primul vector prin selection sort\n");
    printf ("6. Sortati primul vector prin heap sort\n");
    printf ("7. Sortati al doilea vector prin selection sort\n");
    printf ("8. Sort ati al doilea vector prin heap sort\n");
    printf ("9. Iesiti din program\n");
    printf ("Introduceti optiunea: ");
    scanf ("%d", &optiune);
    return optiune;
}
```

```
int main () {
    int n1, n2;
    int v1[100], v2[100];
    int optiune;
    do {
        optiune = meniu ();
        switch (optiune){
            case 1: printf ("Introduceti lungimea primului vector: ");
                    scanf ("%d", &n1);
                      citire_vector (v1, n1);
                      break;
              case 2: printf ("Introduceti lungimea celui de-al doilea vector: ");
                      scanf ("%d", &n2);
                      citire_vector (v2, n2);
                    break;
            case 3: afisare_vector (v1, n1);
                      break;
              case 4: afisare_vector (v2, n2);
                      break;
              case 5: selection_sort (v1, n1);
                     printf ("Vectorul sortat prin selection sort este:\n");
                      afisare_vector (v1, n1);
                      break;
              case 6: heap_sort (v1, n1);
                      printf ("Vectorul sortat prin heap sort este:\n");
                      afisare_vector (v1, n1);
                      break;
              case 7: selection_sort (v2, n2);
                      printf ("Vectorul sortat prin selection sort este:\n");
                      afisare_vector (v2, n2);
                      break;
              case 8: heap_sort (v2, n2);
                      printf ("Vectorul sortat prin heap sort este:\n");
                      afisare_vector (v2, n2);
                      break;
              case 9: printf ("La revedere!\n");
                      break;
```

```
                default: printf ("Optiune invalida!\n");
                        break;
            }
        } while (optiune != 9);
    return 0;
    }
```

Shell Sort

```
#include <stdio.h>
/* function to implement shellSort */
int shell(int a[], int n) {
   /* Rearrange the array elements at n/2, n/4, ..., 1 intervals */
   for (int interval = n/2; interval > 0; interval /= 2)  {
      for (int i = interval; i < n; i += 1)  {
         /* store a[i] to the variable temp and make the ith position empty */
         int temp = a[i];
         int j;
         for (j = i; j >= interval && a[j - interval] > temp; j -= interval)
            a[j] = a[j - interval];
         // put temp (the original a[i]) in its correct position
         a[j] = temp;
      }
   }
   return 0;
}
void printArr(int a[], int n) {
   int i;
   for (i = 0; i < n; i++)
      printf("%d ", a[i]);
}
int main()  {
   int a[] = { 33, 31, 40, 8, 12, 17, 25, 42 };
   int n = sizeof(a) / sizeof(a[0]);
   printf("Before sorting array elements are: \n");
   printArr(a, n);
   shell(a, n);
   printf("\nAfter applying shell sort, the array elements are - \n");
   printArr(a, n);
   return 0;
}
```