Algoritmi za sortiranje

Vežbe

Najčešće metode sortiranja

- Metoda umetanja (*Insertion sort*)
- Metoda zamene suseda (*Bubble sort*)
- Metoda izbora (*Selection sort*)
- Sortiranje po lancima ekvidistantnih zapisa (*Shell sort*)
- Sortiranje po metodi kupa (*Heapsort*)
- Particijska metoda sortiranja (*Quicksort*)
- Sortiranje spajanjem *(Merge sort)*

Bubble sort

- Poredi vrednosti dva susedna elementa i vrši zamenu ukoliko je potrebno. Algoritam ponavlja ovaj proces sve dok ne prođe kroz ceo niz a da ne izvrši ni jednu zamenu. Na ovaj način veće vrednosti “isplivaju” na kraj niza a manje vrednosti “potonu” na početak niza.
- **za:** jednostavnost i lakoća primene.
  protiv: velika neefikasnost.

Bubble sort algoritam

```
void bubbleSort(int numbers[ ], int array_size)
{
 int i, j, temp;
 for (i = (array_size - 1); i >= 0; i--)
   {
        for (j = 1; j <= i; j++)
        {
        if (numbers[j-1] > numbers[j])
        {
        temp = numbers[j-1];
        numbers[j-1] = numbers[j];
         numbers[j] = temp;
         }
        }
   }
}
```

Insertion sort

- Ubacuje svaki element na njegovo odgovarajuće mesto u finalnom nizu. Najjednostavnija primena zahteva dva niza – izvorni niz i niz u koji se element ubacuje. Kako bi se sačuvala memorija, najčešće se radi tako da element koji se trenutno pomera postavlja iza već sortiranih elemenata i ponavlja se zamena dok se ne postavi na odgovarajuću poziciju.
- **za:** relativno jednostavan i lak za primenu.
  protiv: neefikasan za velike nizove.

Insertion sort algoritam

```
void insertionSort(int numbers[], int array_size)
{
int i, j, index;
for (i=1; i < array_size; i++)
    {
   index = numbers[i];
   j = i;
   while ((j > 0) && (numbers[j-1] > index))
    {
   numbers[j] = numbers[j-1];
    j = j - 1;
    }
   numbers[j] = index;
    }
}
```

Selection sort

- Radi tako da bira najmanji preostali nesortirani element u nizu, zatim ga menja sa elementom na sledećoj poziciji koja treba da se popuni.
- **za:** jednostavan i lak za primenu.
  protiv: neefikasan za velike nizove.

Selection sort algoritam

```
void selectionSort(int numbers[], int array_size)
{
int i, j;
int min, temp;
for (i = 0; i < array_size-1; i++){
   min = i;
   // min ce cuvati indeks elementa koji je najmanji od elemenata
   // sa indeksima od i do array_size-1
   for (j = i+1; j < array_size; j++) {
      if (numbers[j] < numbers[min]) min = j;
   }
   // zamena mesta elementima sa indeksima i i min
   temp = numbers[i];
   numbers[i] = numbers[min];
   numbers[min] = temp;
}
}
```

Shell sort

- (Donald Shell, 1959.)
- Nekoliko puta prolazi kroz niz, sortirajući svaki put nizove istih dimenzija koristeći *insertion sort*. Svaki put prolaskom kroz niz uvećava se niz koji treba sortirati dok to ne bude ceo niz.
- **za:** efikasan za nizove srednje veličine.
  protiv: donekle složen algoritam.

Shell sort algoritam

```
void shellSort(int numbers[], int array_size)
 {
 int i, j, increment, temp;
increment = 3;
while (increment > 0)
    {
    for (i=0; i < array_size; i++)
    {
          j = i; temp = numbers[i];
           while ((j >= increment) && (numbers[j-increment] > temp))
           {
           numbers[j] = numbers[j - increment];
          j = j - increment;
          }
          numbers[j] = temp;
     }
    if (increment/2 != 0)
    increment = increment/2;
    else if (increment == 1)
    increment = 0;
    else increment = 1;
     }
}
```

Quick sort

- Algoritam vrlo jednostavan u teoriji, ali veoma je teško pretočiti ga u kod.
- Rekurzivni algoritam čine četiri koraka:
  - Ako niz ima jedan ili manje elemenata za soritiranje, kraj.
  - Odaberi jedan element u nizu za pivot (za to se najčešće koristi krajnje levi element niza).
  - Podeli niz na dva dela – u jednom su elementi veći od pivota a u drugom elementi manji od pivota.
  - Rekurzivno ponavljaj algoritam za obe polovine originalnog niza.
- **za:** izuzetno brz.
  protiv: veoma složen algoritam, rekurzija.

Quick sort algoritam

```
void quickSort(int numbers[], int array_size)
 {
q_sort(numbers, 0, array_size - 1);
 }
 void q_sort(int numbers[], int left, int right)
 {
 int pivot, l_hold, r_hold;
l_hold = left; r_hold = right; pivot = numbers[left];
 while (left < right) {
 while ((numbers[right] >= pivot) && (left < right))
right--;
if (left != right) {
 numbers[left] = numbers[right]; left++; }
while ((numbers[left] <= pivot) && (left < right))
left++;
```

Quick sort algoritam

```
if (left != right) {
numbers[right] = numbers[left];
right--;
}
}
numbers[left] = pivot;
 pivot = left;
left = l_hold;
right  = r_hold;
 if (left < pivot)
q_sort(numbers, left, pivot-1);
if (right > pivot)
q_sort(numbers, pivot+1, right);
 }
```

Merge sort

- Rekurzivni algoritam:
  - U svakom rekurzivnom pozivu ono što želimo da sortiramo delimo na dva dela.
  - Nakon sortiranja obe polovine one se spajaju.
  - Za spajanje se koristi pomoćni niz.
- **za:** izuzetno brz.
  protiv: koristi dodatni memorijski prostor za pomoćni niz, rekurzija.

Merge sort algoritam

```
void mergeSort(int a[], int temp[], int begin, int end){

    // sortiramo niz a pocevsi od indeksa begin do indeksa end

    if(begin >= end)
          return;

    int mid = begin+(end - begin)/2; // mid je indeks srednjeg elementa

    mergeSort(a, temp, begin, mid);
    mergeSort(a, temp, mid+1, end);

    merge(a, temp, begin, mid, end); // spajanje sortiranih polovina
    // jedna polovina ide od begin do mid, druga od mid+1 do end

}
```

Merge sort algoritam

```
void merge(int a[], int temp[], int begin, int mid, int end){

    int i, j, k;
    for(i=begin; i<=end; i++){ // prepisujemo elemente u pomocni niz temp
       temp[i] = a[i];
    }

    i = begin; // indeks za elemente iz prve polovine
    j = mid+1; // indeks za elemente iz druge polovine
    // obe polovine su sortirane (ali nezavisno jedna od druge)
    // prepisujemo elemente iz temp u a, ali tako da budu sortirani
    for(k = begin; k<=end; k++){
       if(i>mid) a[k] = temp[j++]; //  ako smo presli elemente iz prve pol.
       else if(j>end) a[k] = temp[i++]; //ako smo presli el. iz druge pol.
       // u suprotnom poredimo elemente iz obe polovine i manji ide u a
       else if(temp[i] < temp[j]) a[k] = temp[i++];
       else a[k] = temp[j++];
    }
}
```