Zadaci za vježbu za 2. međuispit

Algoritmi sortiranja

Zadatak 1.
Zadan je niz brojeva: **5**, **4**, **9**, **1**, **7**, **10**, **8**, **2**, **3**, **6**. Ilustrirati sortiranje zadanog niza brojeva postupkom ***quicksor****t* tako da se kao stože odabire:

a) prvi element u polju
b) zadnji element u polju

Obavezno ispisati sadržaj polja nakon svake zamjene dvaju elemenata. Nije dopušteno koristiti drugi algoritam za polja manja od neke zadane veličine (*cutoff*, npr. 3 ili 5).

Zadatak 2.
Ilustrirati korake **uzlaznog quicksorta** za podatke: **9, 10, 8, 6, 1, 4, 3, 5, 2, 7**. Stožer odabrati metodom **medijana**. Ispisati izgled polja nakon svake zamjene dvaju elemenata polja.

Zadatak 3.
Zadano je polje brojeva s elementima: **6, 1, 10, 3, 2, 8, 4, 7, 9, 5**. Ilustrirati (ispisati polje nakon svake promjene) sortiranje zadanog niza brojeva algoritmom **shellsort** za niz koraka

a) {4, 3, 1}
b) {3, 2, 1}

Zadatak 4.
Napisati funkciju koja će zadano polje cijelih brojeva (**int**) sortirati algoritmom **bubble sort** (bilo kojom varijantom algoritma). Funkcija treba imati prototip:

```
void bubble(int *polje, int N);
```

Zadatak 5.
Napisati funkciju koja će od dva zadana **sortirana** niza cijelih brojeva, napraviti treći **sortirani** niz koji sadrži sve elemente iz prva dva niza. Dva zadana niza moraju ostati nepromijenjena. Memoriju za treći niz potrebno je zauzeti u funkciji i preko imena (**return**) vratiti u glavni program. Funkcija treba imati prototip:

```
int *merge(int *niz1, int N1, int *niz2, int N2, int *NMerge);
```

Preko argumenta `NMerge`, u glavni program treba vratiti broj elemenata trećeg niza (N1 + N2).

*Primjer: Neka je prvi niz: **1**, **3**, **5**, **6**, a drugi niz: **2**, **4**, **5**, **7.** Treći niz mora biti: **1**, **2**, **3**, **4**, **5**, **5**, **6**, **7** i preko varijable `NMerge` treba vratiti vrijednost 8.*

Stog – objektno orijentirana implementacija[1]

Zajednički dio za sve zadatke sa objektno orijentirano implementiranim stogom :

Za spremanje cijelih brojeva na stog definirana je klasa `Stog` koja ima jedan konstruktor i tri javne funkcije:

```
Stog::Stog();
void Stog::Stavi(int element);
void Stog::Skini(int *element);
int Stog::Prazan();
```

Funkcija `Prazan` vraća 1 ukoliko je stog prazan, a 0 inače. Za klasu `Stog` nije definiran copy konstruktor, a možete pretpostaviti da u svakom objektu klase Stog ima dovoljno mjesta za dodavanje novih elemenata za stog. Pokušaj skidanja elementa iz praznog stoga, dovest će do pogreške. Ukoliko drugačije nije navedeno, smijete koristiti pomoćne stogove, ali ne i dodatna polja.

Zadatak 1:
Napisati funkciju koja će napraviti i vratiti duplikat zadanog stoga (isti takav stog, isti elementi koji imaju isti redoslijed). Zadani stog mora na kraju ostati nepromijenjen. Funkcija mora imati prototip:

```
Stog *duplikat(Stog *zadani);
```

Zadatak 2:
Napisati funkciju koja će iz stoga izbaciti sve parne brojeve. Međusobni redoslijed neparnih brojeva mora ostati nepromijenjen. Funkcija vraća broj izbačenih brojeva te ima sljedeći prototip:

```
int izbaci_parne(Stog *stog);
```

Zadatak 3:
Napisati funkciju koja će stvoriti novi stog koji će sadržavati sve one brojeve iz zadanog stoga za koje funkcija čiji je prototip `int provjeri(int n)` vrati istinu. Poredak prostih brojeva u novom stogu nije bitan. Zadani stog mora na kraju ostati nepromijenjen. Funkcija mora imati prototip:

```
Stog *kopiraj_provjerene(Stog *zadani);
```

Zadatak 4:
Napisati funkciju koja će na osnovu ulaznih stogova `stog1` i `stog2` stvoriti novi stog koristeći pritom sljedeće pravilo da se za novi element u novom stogu uvijek odabere manji od elemenata na vrhu stogova `stog1` i `stog2`. Ukoliko se jedan od stogova isprazni, onda uzeti preostale elemente iz drugog.
Ulazni stogovi moraju nakon završetka funkcije ostati nepromijenjeni. Funkcija mora imati prototip:

```
Stog *spoji(Stog *stog1, Stog *stog2);
```

[1] Implementacija stoga može biti i drugačija pri čemu se rješenja djelomično mijenjaju. Vidi napomenu iza rješenja zadataka sa stogom.

Zadatak 5*:
Napisati funkciju koja će iz stoga izbaciti sve one brojeve koji se pojavljuju dva ili više puta. Funkcija ima sljedeći prototip:

```
void izbaci_duple(Stog *stog);
```

Zadatak 6:
Napisati funkciju koja će iz stoga izbaciti sva pojavljivanja broja n. Funkcija, mora u novi stog, koji je inicijalno prazan, ubaciti brojeve koji predstavljaju koliko je broj bio udaljen od originalnog vrha stoga. Funkcija ima sljedeći prototip:

```
void izbaci_broj(Stog *stog, int n, Stog *novi);
```

Primjer: Ukoliko je ulazni stog bio (vrh je naveden krajnje desno) bio:

```
8 1 4 9 9 2 5 9 6 7 9
```

i ukoliko je funkcija pozvana s `izbaci_broj(stog, 9, novi)`
nakon završetka funkcije, početni stog će biti

```
8 1 4 2 5 6 7
```

a novi stog će imati vrijednosti `0 3 6 7` (može i `7 6 3 0`)

Rješenja

Algoritmi sortiranja

Zadatak 1

a) **5** – stožer
5 – element koji se zamjenjuje

5 4 **9** 1 7 10 8 2 **3** 6
5 4 3 1 **7** 10 8 **2** 9 6
5 4 3 1 **2** **10** 8 7 9 6
2 **4** 3 **1** 5 10 8 7 9 6
2 **1** **3** 4 5 10 8 7 9 6
1 2 3 4 5 **10** 8 7 9 **6**
1 2 3 4 5 6 **8** **7** **9** 10
1 2 3 4 5 6 7 8 9 10

b) **5** – stožer
5 – element koji se zamjenjuje

5 4 **9** 1 7 10 8 2 **3** **6**
5 4 3 1 **7** 10 8 **2** 9 **6**
5 4 3 1 **2** **10** 8 7 9 **6**
5 4 3 **1** **2** 6 8 7 9 10
1 **4** 3 5 **2** 6 8 7 9 10
1 2 **3** **5** **4** 6 8 7 9 10
1 2 3 4 5 **6** **8** **7** 9 10
1 2 3 4 5 6 7 8 9 10

Zadatak 2.

5 – element kandidat za stožer
5 – stožer
5 – element koji se zamjenjuje

9 10 8 6 **1** 4 3 5 2 **7**
1 10 8 6 **7** 4 3 5 2 **9**
1 **10** 8 6 2 4 3 **5** **7** 9
1 5 **8** 6 2 4 **3** 10 **7** 9
1 5 3 6 2 **4** **8** 10 **7** 9
1 5 **3** 6 2 **4** 7 10 8 9
1 **5** **2** 6 **3** 4 7 10 8 9
1 **2** **5** 6 **3** 4 7 10 8 9
1 2 3 **6** **5** **4** 7 10 8 9
1 2 3 4 5 6 7 **10** **8** **9**
1 2 3 4 5 6 7 8 9 10

Zadatak 3.

a) **6** – element koji se zamjenjuje

6 1 10 3 **2** 8 4 7 9 5 k=4
2 1 **10** 3 6 8 **4** 7 9 5
2 1 4 3 6 **8** 10 7 9 **5**
2 1 4 3 6 5 **10** 7 9 **8** k=3
2 **1** 4 3 6 5 8 7 9 10 k=1
1 2 **4** **3** 6 5 8 7 9 10
1 2 3 4 **6** **5** 8 7 9 10
1 2 3 4 5 6 **8** **7** 9 10
1 2 3 4 5 6 7 8 9 10

b) **6** – element koji se zamjenjuje

6 1 10 **3** 2 8 4 7 9 5 k=3
3 1 **10** 6 2 **8** 4 7 9 5
3 1 8 **6** 2 10 **4** 7 9 5
3 1 8 4 2 **10** 6 7 **9** 5
3 1 8 4 2 9 **6** 7 10 **5**
3 1 **8** 4 **2** 9 5 7 10 6 k=2
3 1 **2** 4 8 9 5 7 10 6
2 1 3 4 **8** 9 **5** 7 10 6
2 1 3 4 5 **9** 8 **7** 10 6
2 1 3 4 5 7 8 **9** 10 **6**
2 1 3 4 5 **7** 8 **6** 10 9
2 **1** 3 4 5 6 8 7 10 9 k=1
1 2 3 4 5 6 **8** **7** 10 9
1 2 3 4 5 6 7 8 **10** **9**
1 2 3 4 5 6 7 8 9 10

Zadatak 4.

```
void bubble(int *polje, int N){
	int i, j;
	int pom;
	for (i = 0; i < N-1; i++) {
		for (j = 0; j < N-1-i; j++) {
			if (polje[j+1] < polje[j]){
				pom = polje[j];
				polje[j] = polje[j+1];
				polje[j+1] = pom;
			}
		}
	}
}
```

Zadatak 5.

```
int *merge(int *niz1, int N1, int *niz2, int N2, int *NMerge) {
	int i,j,k;
	int *niz3;

	i = j = k = 0;
	*NMerge = N1 + N2;

	niz3 = (int *)malloc(*NMerge * sizeof(int));
	if (niz3 == NULL) return NULL;

	while (i<N1 && j<N2) {
		if (niz1[i] < niz2[j]) {
			niz3[k] = niz1[i];
			i++;
			k++;
		}
		else {
			niz3[k] = niz2[j];
			j++;
			k++;
		}
	}
	while (i<N1) {
		niz3[k] = niz1[i];
		i++;
		k++;
	}
	while (j<N2) {
		niz3[k] = niz2[j];
		j++;
		k++;

	}

	return N1+N2;
}
```

Stog

Zadatak 1:

```
Stog *duplikat(Stog *zadani){
	Stog *novi = new Stog();
	Stog pomocni;int element;
	while(!zadani->Prazan()){
		zadani->Skini(&element);
		pomocni.Stavi(element);
	}
	while(!pomocni.Prazan()){
		pomocni.Skini(&element);
		zadani->Stavi(element);
		novi->Stavi(element);
	}
	return novi;
}
```

Zadatak 2:

```
int izbaci_parne(Stog *stog){
	Stog pomocni;int element, br=0;
	while(!stog->Prazan()){
		stog->Skini(&element);
		if (element % 2)
			pomocni.Stavi(element);
		else
			br++;
	}
	while(!pomocni.Prazan()){
		pomocni.Skini(&element);
		stog->Stavi(element);
	}
	return br;
}
```

Zadatak 3:

```
Stog *kopiraj_provjerene(Stog *zadani){
	Stog *novi = new Stog();
	Stog pomocni;int element;
	while(!zadani->Prazan()){
		zadani->Skini(&element);
		pomocni.Stavi(element);
	}
	while(!pomocni.Prazan()){
		pomocni.Skini(&element);
		zadani->Stavi(element);
		if (provjeri(element))
			novi->Stavi(element);
	}
	return novi;
}
```

Zadatak 4:

```
Stog *spoji(Stog *stog1, Stog *stog2){
	int el1, el2;
	Stog pom1, pom2, *novi;
	novi = new Stog();
	while(!stog1->Prazan() && !stog2->Prazan()){
		stog1->Skini(&el1);
		stog2->Skini(&el2);
		if (el1 < el2){
			novi->Stavi(el1);
			pom1.Stavi(el1);
			//vrati većeg(neiskorištenog) na njegov stog,
			//tako da opet bude na vrhu
			stog2->Stavi(el2);
		}
		else{
			novi->Stavi(el2);
			pom2.Stavi(el2);
			stog1->Stavi(el1);
		}
	}
	while(!stog1->Prazan()){
		stog1->Skini(&el1);
		novi->Stavi(el1);
		pom1.Stavi(el1);
	}
	while(!stog2->Prazan()){
		stog2->Skini(&el2);
		novi->Stavi(el2);
		pom2.Stavi(el2);
	}
	while(!pom1.Prazan()){
		pom1.Skini(&el1);
		stog1->Stavi(el1);
	}
	while(!pom2.Prazan()){
		pom2.Skini(&el2);
		stog2->Stavi(el2);
	}
	return novi;
}
```

Zadatak 5*:

Ideja: Uzeti prvi element na stogu i spremiti ga u x. Zatim ga uspoređivati sa preostalim elementima u stogu. Ukoliko pojedini element nije jednak x sačuvati ga privremeno u pomoćnom stogu `pom1`*. Ukoliko je taj element jednak x, onda označiti da x ima duplikate i taj broj odbaciti (ne čuvati ga u pomoćnom stogu). Nakon prolaska kroz sve elemente stoga, vrijednost duplikat će označavati da li x ima duplikata ili ne. Ako nije imao duplikata, onda x sačuvati premještanje u pomoćni stog* `pom2`*. Sve elemente iz* `pom1` *vratiti u originalni stog i ponoviti postupak. Nakon što postupak bude gotov, tj. na*

originalnom stogu više ne bude nikakvih elemenata, na stogu `pom2` će se nalaziti samo oni elementi koji nisu imali duplikata u stogu te ih treba vratiti na originalni stog.

```
void izbaci_duple(Stog *stog){
	Stog pom1, pom2;int duplikat=1, x, element;
	while(!stog->Prazan()){
		stog->Skini(&x);
		duplikat = 0;
		while(!stog->Prazan()){
			stog->Skini(&element);
			if (x == element)
				duplikat = 1;
			else
				pom1.Stavi(element);
		}
		if (!duplikat)
			pom2.Stavi(x);
		while(!pom1.Prazan()){
			pom1.Skini(&element);
			stog->Stavi(element);
		}
	}
	while(!pom2.Prazan()){
		pom2.Skini(&element);
		stog->Stavi(element);
	}
}
```

Zadatak 6:

```
void izbaci_broj(Stog *stog, int n, Stog *novi){
	Stog pom;int element, udaljenost=0;
	while(!stog->Prazan()){
		stog->Skini(&element);
		if (element == n)
			novi->Stavi(udaljenost);
		else
			pom.Stavi(element);
		udaljenost++;
	}
	while(!pom.Prazan()){
		pom.Skini(&element);
		stog->Stavi(element);
	}
}
```

Napomena:
Implentacija stoga je mogla biti ponešto drugačija. Primjerice, mogla je izgledati ovako:

```
Stog::Stog();
int Stog::Stavi(int element);
int Stog::Skini(int *element);
```

pri čemu bi funkcija Skini vraćala 1 ukoliko je skidanje elementa sa stoga uspjelo, a 0 ako skidanje nije uspjelo (npr. iz razloga jer je stog bio prazan)
U tom slučaju rješenja zadataka bi bila ponešto drugačija, pa bi primjerice zadatak 6. imao sljedeće rješenje:

```
void izbaci_broj(Stog *stog, int n, Stog *novi){
	Stog pom;int element, udaljenost=0;
	while(stog->Skini(&element)){
		if (element == n)
			novi->Stavi(udaljenost);
		else
			pom.Stavi(element);
		udaljenost++;
	}
	while(pom.Skini(&element)){
		stog->Stavi(element);
	}
}
```