Elektrotehnički fakultet Sarajevo
Odsjek za računarstvo i informatiku
Algoritmi i strukture podataka

Analiza vremenske kompleksnosti algoritama

Dodatni materijal za vježbu 1

U tabeli 1 su navedeni primjeri koda i njihova vremenska kompleksnost u Big - O notaciji.

Primjer	Vrijeme izvršenja	Aproks.	Big-O	Naziv (kompleksnost)
`z=x+y;`	k	1	O(1)	konstantna

`for (i=0; i<n; i++)`
`  if (x[i]>max)`
`  max=x[i];`
	$k_1n + k_2$	n	O(n)	linearna

`for (i=0; i<n; i++)`
`  for (j=0; j<i; j++)`
`  if (x[i]>x[j])`
`  swap(x[i],x[j]);`
	$k_1n^2 + k_2n + k_3$	n^2	O(n^2)	kvadratna

`for ... for ... for ...`	$k_an^a + ... + k_2n^2 + k_1n+k_0$	n^a	O(n^a)	polinomska
`for (i=1; i<n; i*=2)` `  sum += niz[i];`	$k_1 \log_2 n + k_2$	log n	O(log n)	logaritamska

`for (i=0; i<n; i++)`
`  for (j=1; j<n; j*=2)`
`  if (y[j]+x[i]==...)`
	$k_1 n \log_2 n + k_2 n + k_3$	n log n	O(n log n)	linearitamska

`int Fib(int n) {`
`  if (n<=1) return 1;`
`  return Fib(n-1) + Fib(n-2);`
`}`
	...	2^n	O(2^n)	eksponencijalna

sve permutacije niza		n!	O(n!)	faktorijelna

Tabela 1.

Zauzeće memorije

Koliko različiti C++ tipovi podataka zauzimaju memorije? Za savremene PC računare važi:

Tip	Memorija u bajtima (byte)
bool	1
char	1
int	4
long	4 ili 8 (4 na svim Windows-ima, 8 na 64-bitnim Linux-ima)
long long	8
float	4
double	8
tip*	4 ili 8 (ovisno da li je računar[1] 32-bitni ili 64-bitni)
struct	zbir veličina elemenata
tip niz[N]	N * sizeof(tip)
vector<tip>	minimalno: N * sizeof(tip) + 3 * sizeof(void*)
lista<tip>	N * sizeof(cvor) tipično: N * (sizeof(tip) + 2*sizeof(void*))

Veličina objekta:
- veličina svih ne-statičkih atributa (privatnih i javnih),
- uključujući i atribute klase roditelja,
- po jedan pokazivač (4 ili 8 bajta) za svaku virtuelnu metodu,
- treba voditi računa o poravnanju (npr. char atribut u nekim situacijama može zauzeti 4 bajta).

Primjer 1

Data je sljedeća klasa:

```
class Niz {
      int* niz;
      int brojElemenata, kapacitet;
public:
      Niz() : niz(new int[1000]), brojElemenata(0), kapacitet(1000) {}
      …
};
```

Kakvo je njeno zauzeće memorije?

[1] Preciznije, i računar i kompajler moraju biti 64-bitni da bi pokazivač bio 64-bitni. Na primjer, podrazumijevana instalacija Code::Blocks-a na 64-bitni Windows koristi 32-bitni kompajler.

Rješenje.

- Atribut `int* niz` - pokazivač - 4 ili 8 bajta
- Atributi `int brojElemenata, kapacitet` - int - po 4 bajta.
- Dinamički je alocirano 1000 elemenata tipa `int`, što ukupno daje 4000 bajta.
- Klasa nema metode za promjenu veličine alocirane memorije.

Ukupno: 4012 ili 4016 bajta.

Zadaci za vježbu

Odrediti vremensku kompleksnost sljedećih algoritama u najboljem i u najgorem slučaju.

1.
```
int f(1), sum(0);
for (int i(2); i<=n; i++)
      f *= i;
for (int i(0); i<f; i++)
      sum += i;
```

Rješenje: O(n!) - f je faktorijel od n, a druga petlja se izvršava f puta

2.
```
bool prost(int x) {
      bool prost(true);
      for (int i(2); i<x/2; i++)
            if (x%i==0) prost=false;
      return prost;
}
for (int i=2; i<n; i++)
      if (prost(i)) sum += i;
```

Rješenje: $O(n^2)$ - funkcija prost je O(n) a poziva se za svaki broj od 2 do n

3.
```
bool prost(int x) {
      for (int i(2); i<x/2; i++)
            if (x%i==0) return false;
      return true;
}
for (int i=2; i<n; i++) {
      if (prost(i)) break;
      sum += i;
}
```

Rješenje: O(1) - 2 je prost broj pa će se petlja odmah prekinuti

4.
```
int suma(0), max(niz[0]);
for (int i(1); i<n; i++)
      if (niz[i]>max) max=niz[i];
for (int i(0); i<n; i++) {
      suma += niz[i];
      if (suma>max) break;
}
```

Rješenje: O(n) - prva petlja je O(n), druga petlja može u određenim slučajevima brže završiti, ali u najgorem slučaju je opet O(n) tako da ne utiče na ukupnu složenost

5.
```
int suma(0), max(niz[0]);
for (int i(1); i<n; i++)
     if (niz[i]>max) max=niz[i];
for (int i(0); i<n; i++)
     for (int j(0); j<n; j++)
          if (i!=j && niz[i]!=max && niz[j]!=max &&
          niz[i]+niz[j]>max)
               return true;
return false;
```

Rješenje: O(n) u najboljem slučaju, $O(n^2)$ u najgorem slučaju (ako su posljednja dva člana niza znatno veći od ostalih)

6.
```
double stepen(double x, int n) {
     double rez(1);
     for (int i(0); i<n; i++) rez *= x;
     return rez;
}
int fakt(int n) {
     int rez(1);
     for (int i(2); i<=n; i++) rez *= i;
     return rez;
}
double sinus(double x, int n) {
     double suma(0);
     for (int i(1); i<=2*n-1; i+=2)
          suma += stepen(-1,i/2) * stepen(x,i) / fakt(i);
     return suma;
}
```

Rješenje: složenost funkcije sinus je $O(n^2)$ - stepen i fakt su O(n), pozivaju se sekvencijalno unutar petlje

7.
```
double sinus (double x, int n) {
     double rez(0), brojnik(x), nazivnik(1);
     int stepen(1), znak(1);
     do {
          rez += znak*brojnik/nazivnik;
          znak *= -1;
          brojnik *= x*x;
          nazivnik *= (stepen+1)*(stepen+2);
          stepen += 2;
     } while (stepen <= 2*n-1);
     return suma;
}
```

Rješenje: druga varijanta funkcije sinus ima složenost O(n)

9.
```
for (int i(0); i<vel; i++, j=1) {
     if (niz[i]>i) continue;
     while (j<i/3) {
          niz2[j] = niz[i];
          j*=2;
     }
}
```

Rješenje: O(n) u najboljem slučaju, a O(n log n) u najgorem.

```
10.  j=vel/2;
     for (int i(0); i<vel; i++) {
          if (niz[i]>0) continue;
          if (i>j) break;
          sum += niz[j];
          j /= 2;
     }
```

Rješenje: O(n) u najgorem slučaju (svi članovi su pozitivni pa se uvijek izvršava continue), O(log n) u najboljem (ako su svi negativni j će se dijeliti sa dva pa će se petlja završiti za $\log_2$vel koraka)

```
11.  for (int i(1); i<n; i*=2)
          n++;
```

Rješenje: O(log n) – vidijeti primjer iz tabele 1

```
12.  for (int i(0); i<n; i++) {
          double k = sqrt(i);
          if ( fabs(k - int(k)) < 1e-7 ) { // if (k == int(k))
               for (int j(0); j<n; j++)
                    k += niz[j];
          }
     }
```

Rješenje: O($n^{1,5}$) - uslov u if-u je ispravan za svaki puni kvadrat od 0 do n, a takvih je sqrt(n)

```
13.  int sum(0);
     for (int i(0); i<x; i++) {
          for (int j(0); j<x; j++)
               if (i==j) continue;
               if (niz[i]==niz[j]) break;
               sum += niz[j];
          }
     }
```

Rješenje: u najgorem slučaju $O(n^2)$, u najboljem slučaju $O(n)$ (ako su svi članovi niza jednaki)

```
14.  int x(1);
     for (int i(0); i<a; i++)
          x *= 2;
     for (int j(2); j<x; j++)
          sum += 1.0/j;
```

Rješenje: ovisnost o ulaznoj vrijednosti a je $O(2^n)$

```
15.  int max(0), sum(0);
     for (int i(1); i<vel; i++)
          if (niz[i]>niz[max]) max=i;
     for (int i(0); i<max; i++)
          sum += niz[i];
```

Rješenje: O(n) - prva petlja se izvršava vel puta, a druga max puta gdje max iz intevala [0,vel)

```
16.  int maks(int* niz, int vel) {
          int rez(niz[0]);
          for (int i(1); i<vel; i++)
               if (niz[i]>rez) rez=niz[i];
          return rez;
     }
     int min(int* niz, int vel) {
          int rez(niz[0]);
          for (int i(1); i<vel; i++)
               if (niz[i]<rez) rez=niz[i];
          return rez;
     }
     ...
     for (int i(0); i<vel; i++)
          if (maks(niz,i) - min(niz,i) == niz[i]) break;
```

Rješenje: funkcije maks i min su O(n), u petlji se pozivaju sekvencijalno, a uslov ne mora nikada biti ispunjen, dakle u najgorem slučaju $O(n^2)$ a u najboljem O(n)

```
17.  int a(0), b(0), c(0);
     for (a=0; a<vel; a++) {
          for (b=0; b<vel; b++) {
               for (c=0; c<vel; c++) {
                    if (polje[a][b][c] < 0) break;
               }
               if (c<vel) vel=c;
          }
     }
```

Rješenje: u najgorem slučaju $O(n^3)$, u najboljem O(1)

```
18.  int f(1), sum(0);
     for (int i(2); i<=n; i++)
          f *= n;
     for (int i(0); i<n; i++)
          for (int j(0); j<n; j++)
               for (int k(0); k<n; k++)
                    if (niz[i]+niz[j]+niz[k] == f)
                         return true;
     return false;
```

Rješenje: u najgorem slučaju $O(n^3)$, u najboljem O(n) (i dalje se mora izvršiti prva petlja)

```
19.  int f(1), sum(0);
     for (int i(2); i<=n; i++)
          f *= i;
     for (int i(0); i<n; i++)
          for (int j(0); j<n; j++)
               for (int k(0); k<f; k++)
                    if (niz[i]+niz[j] == k)
                         return true;
```

Rješenje: u najgorem slučaju O(n!) - tehnički $O(n^2 * n!)$ ali n! dominira nad n^2 tako da se to zanemaruje; u najboljem O(n)

20.

```
bool prost(int n) {
     bool prost(true);
     for (int i(2); i<n/2; i++)
          if (n%i==0) prost=false;
     return prost;
}
int suma(int n) {
     int s=0;
     for (int i(2); i<n; i++)
          if (prost(i))
               s+=i;
     return s;
}
...
for (int i(2); i<n; i++)
     if (prost(suma(i))
          count++;
```

Ovo je program koji određuje koliko brojeva *i* ima između 1 i *n* takvih da je suma svih prostih brojeva od 1 do *i* i sama prost broj. Funkcija prost ima složenost O(n). U funkciji suma poziva se funkcija prost za sve brojeve od 2 do n. Vrijeme izvršenja je proporcionalno sa $\sum n/2$ što je $\frac{n(n-1)}{4} = \frac{n^2}{4} - \frac{n}{4}$ pa je to $O(n^2)$. U traženom isječku koda najprije se poziva funkcija suma, zatim se rezultat te funkcije šalje funkciji prost. Funkcija suma izračunava sumu svih prostih brojeva od 1 do n. Suma svih *prostih* brojeva od 1 do n je *sigurno manja* od sume *svih* brojeva od 1 do n koja je $O(n^2)$. Dakle složenost poziva funkcije suma dominira nad složenošću poziva funkcije prost. Sve to se nalazi u jednoj for petlji koja se izvršava n puta pa je ukupna složenost isječka koda $O(n^3)$.