# Algoritmi in podatkovne strukture – 2

Urejanje (*sorting*)

osnove, metode deli in vladaj, kopica

# Urejanje

Imamo vhodni niz predmetov: $X = [x_1, x_2, ...x_n]$.

Za poljubna predmeta velja, da sta urejena, kar pomeni, da je eden večji ali enak kot drugi: $x_i \leq x_j$ ali $x_i \leq x_j$ in da je ta relacija tranzitivna.

Rezultat urejevalnega postopka je niz predmetov

$$X' = [x'_i \mid (x'_i = y_j) \text{ in } x'_i \leq x'_{i+1}]$$

# Zahtevnost problema

- o množici $X$ ne vemo nič drugega, kot to, da je urejena
- naš računalnik lahko dela samo primerjave
- niz $X'$ je permutacija niza $X$

# Zahtevnost problema

- urejevalni algoritem lahko v spložnem popišemo kot, da deluje v dveh korakih:
  1. ugotovi kakšna permutacija $X$ je $X'$
  2. permutiraj predmete v $X$
- drugi korak lahko naredimo vedno v času $\Theta(n)$
- ker je $n!$ permutacij niza $X$ in ker je možno, da bo urejevalni algoritem moral narediti katerokoli permutacijo, mora biti tudi vse razpoznati
- permutacije algoritem razlikuje tako, da ubere drugačne vejitve v toku svojega izvajanja – vsaka vejitev določa dva različna toka
- najslabši čas izvajanja algoritma je največje število vejitev v toku izvajanja (iskanja permutacij). Zato:
  - v najboljšem primeru potrebujemo vsaj $n!$ vejitev (ker potrebujemo $n!$ različnih tokov izvajanja)
  - v najboljšem primeru (najboljši možen algoritem) bo imel vejitve razporejene v obliki drevas in višina le-tega je: $\lg n! = n \lg n - O(n)$.

(POIŠČITE NA SPLETU DRUGI ČLEN IZRAZA TER GA OBRAZLOŽITE.)

# Urejanje z vstavljanjem

```
public void insertionSort(int elts[]) {
  int elt, tmp, i, j;
  for (i= 1, i < elts.length()-1, i++) {
    elt= elts[i];
    for (j= 0, (j < i) && (elt > elts[j]), j++);
    for ( , j < i, j++) {
      tmp= elts[j]; elts[j]= elt; elt= tmp
    }
    elts[i]= elt;
  }
}
```

# Časovna zahtevnost

- velikost polja `elts` je $n$
- zunanja zanka se izvede $(n-1)$-krat
- notranji zanki (skupaj) se izvedeta: $1$, $2$, ..., $n-2$ -krat, kar skupaj znese $O(n^2)$ krat
- čas izvajanja je kvadratičen (v najslabšem primeru)

# Metoda *deli in vladaj*

Osnovna ideja:

1. Če je problem majhen, ga reši, sicer
2. problem razdeli na manjše podprobleme,
3. reši vsakega od podproblemov (z isto metodo)
4. združi rešitve podproblemov v rešitev prvotnega problema

# Shema algoritma

```
public solutionClass DivideAndConquer( problem ) {
  if Basic(problem) solution= Solve(problem);
  else {
    problems[]= Split(problem);
    for parallel (i= 0, i < problems.length, i++)
      solutions[i]= DivideAndConquer(problems[i]);
    solution= Combine(solutions)
  }
  return solution;
}
```

# Časovna zahtevnost

Zaporedna inačica:

$$T(n) = \texttt{Split.T} + k \cdot T(n/k) + \texttt{Combine.T} + \texttt{Solve.T}$$

Vzporedna inačica:

$$T(n) = \texttt{Split.T} + max(T(n/k)) + \texttt{Combine.T} + \texttt{Solve.T}$$

# Urejanje z zlivanjem – *merge sort*

- problem je osnoven, če je zaporedje dolgo 1 in ga ni potrebno reševati – $O(1)$
- *Split*: zaporedje dolžine $n$ razdelimo na dve podzaporedji dolžin $\lfloor \frac{n}{2} \rfloor$ in $\lceil \frac{n}{2} \rceil$ $O(1)$
- *Combine*: zlivanje $O(n)$

Skupna časovna zahtevnost?

$$T(n) = O(n + n/2 + n/4 + \ldots) = O(n \log n)$$

(NAPIŠITE ALGORITEM V JAVI.)

# Hitro urejanje – *quick sort*

- problem je osnoven, če je zaporedje dolgo 1 in ga ni potrebno reševati – $O(1)$
- *Split*: zaporedje dolžine $n$ razdelimo na dve podzaporedji, kjer so elementi prvega podzaporedja vsi manjši od elementi drugega podzaporedja $O(n)$
- *Combine*: nič $O(1)$

Skupna časovna zahtevnost?

Tokrat gre lahko vse narobe in je eno podzaporedje dolgo samo en element. Potem dobimo:

$$T(n) = O(n + n - 1 + n - 2 + ...) = O(n^2)$$

(NAPIŠITE ALGORITEM V JAVI.)

# Komentar

- pri urejanju z zlivanjem v najslabšem primeru naredimo manj urejanj kot pri hitrem urejanju *TODA*:
  - ali je samo štetje primerjanj pri resničnem računalniku smiselno?
    Kako izgleda računalnik? Pomnilniška hierarhija.
  - kako pogosto nastopi takšen slab primer pri hitrem urejanju? Kaj če se mu izognemo?
    Način izbire razločilnega elementa.
- kaj potem?

(SPROGRAMIRAJTE OBE METODI UREJANJA IN JU PRIMERJAJTE NA NAKLJUČNEM NABORU ŠTEVILK.)

# Rezultat

| | najboljše | najslabše |
|---|---|---|
| urejanje z vstavljanjem | $O(n^2)$ | $O(n^2)$ |
| urejanje z zlivanjem | $O(n \log n)$ | $O(n \log n)$ |
| hitro urejanje | $O(n \log n)$ | $O(n^2)$ |
| urejanje z mehurčki | ? | ? |

(DOPOLNITE TABELO. DODAJTE ŠE KAKŠNO VPRAŠANJE.)

# Fizikalni pristop

Uporabimo orodja, ki jih imamo v škatli – vrste s prednostjo.

```
int[] Sort(int[] polje) {
  int i;
  PQ pq= new PQ(polje.Size());
  for (i=0; i<polje.Size(); i++) pq.Insert(polje[i]);
  for (i=0; !pq.Empty(); i++) polje[i]= pq.DelMin();
  return polje;
}
```

- časovna zahtevnost: $O(n \log n)$, prostorska zahtevnost: $O(n)$
- kako zmanjšati prostorsko zahtevnost na $n$ – kopično urejanje, *heapsort*

# Zahtevnost

| **urejanje** | **čas** |
|---|---|
| z vstavljanjem | $O(n^2)$ |
| z zlivanjem | $O(n \log n)$ |
| s kopico | $O(n \log n)$ |
| hitro urejanje | $O(n^2)$ |
| z mehurčki | ? |

Vprašanja:

- Povsod je ocena prostora $O(n)$; lahko to oceno izboljšamo na $k \cdot n$?
  - Kakšen je najmanjši *možen* $k$?
  - Kakšen $k$ znamo doseči za posamezno obliko urejanja?
- Povsod ocena časa vsebuje $O$; lahko to oceno izboljšamo?
  - Kakšen je najmanjši *možen* čas?
  - Kakšna je boljša ocena (brez $O$ vsaj pri vodilnem členu)?[1]

[1] Npr.: $7,4n \lg n + O(n)$ ali celo $7,4n \lg n + 3,14n - o(n)$ (številke so izmišljene).