STRUKTURE PODATAKA I ALGORITMI

Predavanje 12

Ishod 5

1

SORTIRANJE

2

Uvod u sortiranje

- Sortiranje (engl. *sorting*) je postupak slaganja niza elemenata u rastući ili padajući redoslijed
- Kod sortiranja je bitno voditi računa o sljedećem:
  - Kriterij sortiranja – po čemu sortiramo i kojim redoslijedom
    - Primjerice, osobe možemo sortirati abecedno po imenu i prezimenu
  - Složenost algoritma (tablica na sljedećem slajdu)
  - Koliko algoritmu treba memorije kako bi izvršio sortiranje
  - Broj zamjena (engl. *swap*)
  - Je li algoritam stabilan
    - Stabilan algoritam ne mijenja međusobni raspored dva jednaka elementa

Algoritmi sortiranja (1/2)

- Među najpoznatijim algoritmima sortiranja su:

Algorithm	Time Complexity		
	Best	Average	Worst
Quicksort	Ω(n log(n))	θ(n log(n))	O(n^2)
Mergesort	Ω(n log(n))	θ(n log(n))	O(n log(n))
Timsort	Ω(n)	θ(n log(n))	O(n log(n))
Heapsort	Ω(n log(n))	θ(n log(n))	O(n log(n))
Bubble Sort	Ω(n)	θ(n^2)	O(n^2)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)
Tree Sort	Ω(n log(n))	θ(n log(n))	O(n^2)
Shell Sort	Ω(n log(n))	θ(n(log(n))^2)	O(n(log(n))^2)
Bucket Sort	Ω(n+k)	θ(n+k)	O(n^2)
Radix Sort	Ω(nk)	θ(nk)	O(nk)
Counting Sort	Ω(n+k)	θ(n+k)	O(n+k)
Cubesort	Ω(n)	θ(n log(n))	O(n log(n))

Preuzeto s bigocheatsheet.com

Algoritmi sortiranja (2/2)

- Algoritmi koje ćemo proučavati u ovom predavanju spadaju u tzv. *comparison sort* algoritme
  - *Comparison sort* algoritam radi na principu uspoređivanja elemenata
    - Mi dajemo odgovor na pitanje: je li element *A* manji od elementa *B*
- Postoje i sortiranja bazirana na nekim drugim tehnikama
  - Najpoznatija je *integer sort* u kojoj se podaci uspoređuju po ključu
    - Moguće postići linearnu složenost najgoreg slučaja

Konvencija

- Na sljedećim slajdovima vrijedi:
  - Elemente koje trebamo sortirati se nalaze u polju koje nazivamo `data`
  - Broj elemenata u polju označavamo s n
  - Indeks označavamo s i
    - Obično ide od 0 do $n - 1$
  - Pretpostavljamo da želimo sortirati cijele brojeve od manjih prema većima
    - Sličan princip vrijedi za sve tipove podataka
    - Sličan princip vrijedi i za sortiranje od većih prema manjim

BOGO SORT

Bogo sort

- Poznat i kao: stupid sort, slowsort, random sort, monkey sort, …
- Izrazito neefikasan, u praksi se nikad ne koristi:
  - Slučajna verzija: nema ograničavajuće funkcije
  - Deterministička verzija: $O((n+1)!)$
- Taktika:
  1. Provjeri jesu li elementi sortirani (ako jesu, kraj) => $\Omega(n)$
  2. Ako nisu, promiješaj elemente slučajnim redoslijedom i idi na prethodni korak
     - Deterministička verzija kaže da ide sljedeća permutacija

Permutacije

- Permutacija je naziv za preslagivanje elemenata u neki drugi redoslijed
  - Primjerice, imamo elemente: [1, 2, 3, 4, 5]
  - Jedna njihova moguća permutacija: [2, 5, 4, 3, 1]
  - Ako imamo *n* elemenata, onda imamo *n*! permutacija
- STL dolazi s nekoliko algoritama koji pojednostavljuju rad s permutacijama
  - `lexicographical_compare` provjerava je li prvi niz leksikografski manji od drugog
  - `next_permutation` za zadani niz izračunava sljedeću permutaciju

ALGEBRA BERNAYS SVEUČILIŠTE

9

Leksikografsko uspoređivanje

- Koristi se za abecedno sortiranje riječi u rječnicima
- Pretpostavimo da uspoređujemo dvije riječi
  - Uspoređuju se znakovi na istim mjestima, počevši od početka
    - Ako uzmemo prva dva znaka koji su različiti, manja riječ je ona čiji znak abecedno dođe prije
    - Ako dođemo do kraja jedne riječi i svi znakovi su bili jednaki, manja riječ je ona s manje znakova
- Primjerice, usporedimo riječi „Mario” i „Marko”
  - „Mario” je manji jer je slovo „i” abecedno prije „k”
- Primjerice, usporedimo riječi „Konjuktivitis” i „Konj”
  - „Konj” je manji jer su prva četiri slova jednaka, a „Konj” je kraća riječ

ALGEBRA BERNAYS SVEUČILIŠTE

10

Funkcije za permutacije (1/2)

- `bool lexicographical_compare(be1, end1, be2, end2)`
  - Vraća `true` ako je prvi raspon leksikografski manji od drugog
  - Primjer:

```
string s1 = "Mario";
string s2 = "Marko";

cout << lexicographical_compare(
            s1.begin(), s1.end(), s2.begin(), s2.end()) << endl;

vector<int> v1 = { 1, 2, 3, 4, 5 };
vector<int> v2 = { 1, 2, 3, 4, 6 };

cout << lexicographical_compare(
            v1.begin(), v1.end(), v2.begin(), v2.end()) << endl;
```


Funkcije za permutacije (2/2)

- `bool next_permutation(begin, end)`
  - Preslaguje raspon u sljedeću veću leksikografsku permutaciju:
    - Vraća `true` ako postoji veća leksikografska permutacija
      - Najveća leksikografska permutacija su elementi sortirani padajuće
    - Inače vraća `false` i elemente preslaguje u najmanju permutaciju
      - Najmanja leksikografska permutacija su element sortirani rastuće
  - Primjerice:

```
vector<int> v({ 5, 3, 1, 2, 4 });
int n = nfact(v.size());
for (int i = 0; i < n; i++) {
    std::cout << v[0] << ' ' << v[1] << ' ' << v[2] << ' ' <<
                                                v[3] << ' ' << v[4];
    cout << " (" << next_permutation(v.begin(), v.end()) << ")"
                                                        << endl;
```

BUBBLE SORT

13

Bubble sort

Bubble Sort Ω(n) Θ(n^2) O(n^2)

- Taktika: usporedi dva susjedna elementa i ako nisu u dobrom redoslijedu, zamijeni ih
  - Nakon prvog prolaska najveći je element završio na kraju polja i više ga ne diramo jer čini **sortirani dio polja**
    - Kažemo da je isplivao (engl. *bubble up*) na kraj polja
  - Ponavljamo proceduru za preostale elemente
  - Algoritam staje nakon $n - 1$ prolaza ili nakon prvog prolaska bez zamjene

14

Performanse

- Najlakši za implementirati, ali zato ima najlošiju efikasnost
  - U najgorem, ali i u prosječnom slučaju je $O(n^2)$
  - Problem je i velik broj zamjena mjesta
- Jedina prednost je ugrađena sposobnost detekcije da je polje već sortirano
  - Nema bržeg algoritma u najboljem slučaju
- Ne koristi se u praksi

ALGEBRA BERNAYS SVEUČILIŠTE

15

Bubble sort primjer

- Primjer rada:

```
   Start: [16, 32, 42, 10, 20]
Prolaz 1: [16, 32, 10, 20, 42]
Prolaz 2: [16, 10, 20, 32, 42]
Prolaz 3: [10, 16, 20, 32, 42]
Prolaz 4: [10, 16, 20, 32, 42]
    Kraj: [10, 16, 20, 32, 42]
```

- Resursi:
  - cs.usfca.edu/~galles/visualization/ComparisonSort.html
  - youtube.com/watch?v=lyZQPjUT5B4

ALGEBRA BERNAYS SVEUČILIŠTE

16

Bubble sort implementacija

```
for (int i = 0; i < n - 1; i++) { // Zadnjeg ne uspoređujemo.
    bool sortirano = true;

    for (int j = 0; j < n - 1 - i; j++) { // Ostatak je sortiran.
        if (data[j] > data[j + 1]) {
            swap(data[j], data[j + 1]);
            sortirano = false;
        }
    }

    if (sortirano) {
        break;
    }
}
```


SELECTION SORT

Selection sort

Selection Sort Ω(n^2) Θ(n^2) O(n^2)

- Taktika:
  - Neka polje ima sortirani i nesortirani dio
  - Pronađi najmanji element u nesortiranom dijelu:
    - Zamijeni ga s prvim elementom u nesortiranom dijelu
    - Proglasi da sad taj element pripada sortiranom dijelu
  - Ponavljaj dok cijelo polje ne bude sortirano

Selection sort primjer

- Primjer rada:

```
   Start: [16, 32, 42, 10, 20]
Prolaz 1: [10, 32, 42, 16, 20]
Prolaz 2: [10, 16, 42, 32, 20]
Prolaz 3: [10, 16, 20, 32, 42]
Prolaz 4: [10, 16, 20, 32, 42]
    Kraj: [10, 16, 20, 32, 42]
```

- Pogledati primjer na cs.usfca.edu/~galles/visualization/ComparisonSort.html

Selection sort implementacija

```
for (int i = 0; i < n - 1; i++) { // Nesortirani dio
    int min_index = i;
    for (int j = i + 1; j < n; j++) {
        if (data[j] < data[min_index]) {
            min_index = j;
        }
    }
    swap(data[min_index], data[i]);
}
```


INSERTION SORT

Insertion sort

- Taktika (polje ima sortirani i nesortirani dio):
  - Uzmi prvi element u nesortiranom dijelu:
    - Proglasi da sad taj element pripada sortiranom dijelu
    - Mijenjaj mu mjesto s elementima u sortiranom dijelu dok ne dođe na ispravno mjesto
  - Ponavljaj dok cijelo polje ne bude sortirano

23

Insertion sort primjer

- Primjer rada:

```
   Start: [16, 32, 42, 10, 20]
Prolaz 1: [16, 32, 42, 10, 20]
Prolaz 2: [16, 32, 42, 10, 20]
Prolaz 3: [10, 16, 32, 42, 20]
Prolaz 4: [10, 16, 20, 32, 42]
    Kraj: [10, 16, 20, 32, 42]
```

- Pogledati primjer na cs.usfca.edu/~galles/visualization/ComparisonSort.html

24

Insertion sort implementacija

```
for (int i = 1; i < n; i++) { // Prvog preskačemo.
    for (int j = i; j > 0 && data[j - 1] > data[j]; j--) {
        swap(data[j], data[j - 1]);
    }
}
```


25

SHELL SORT

26

Shell sort

Shell Sort Ω(n log(n)) Θ(n(log(n))^2) O(n(log(n))^2)

- Nazvan po autoru, Donaldu Shellu
- Modifikacija insertion sorta
  - Rješava problem insertion sorta: mali elementi pri kraju polja zahtijevaju puno zamjena mjesta
- Taktika: polje ćemo prirediti tako da insertion sort na njemu radi brže. Prirađivanje se radi sortiranjem manjih dijelova polja (s ciljem da se neki ekstremni elementi prije dovedu do svog mjesta nego što bi se to desilo insertion sortom)
- Efikasnost ovisi o algoritmu odabira manjih dijelova polja

ALGEBRA BERNAYS SVEUČILIŠTE

27

Detaljniji opis

- Detaljniji opis shell sorta:
  - Dijelimo polje na h_1 potpolja, a svako potpolje čine sljedeći elementi:
    - 1. potpolje: `data`[0], `data`[h_1], `data`[$2h_1$], …
    - 2. potpolje: `data`[1], `data`[h_1+1], `data`[$2h_1$+1], …
    - h_1-to potpolje: `data`[h_1-1], `data`[$2h_1$-1], `data`[$3h_1$-1], …
  - Svako potpolje sortiramo insertion sortom
  - Uzimamo broj $h_2 < h_1$ i ponavljamo gornji postupak sve dok ne dođemo do $h_k = 1$
  - Niz $h_1, h_2, \ldots, h_k = 1$ nazivamo sekvenca razmaka (engl. *gap sequence*)
    - Sekvenca $h_k = 1$ predstavlja insertion sort cijelog polja

ALGEBRA BERNAYS SVEUČILIŠTE

28

Shell sort primjer

- Pogledati primjer na cs.usfca.edu/~galles/visualization/ComparisonSort.html
- Imamo polje od 50 elemenata
- $h_1 = 25$ (dakle, u prvom prolazu imamo 25 potpolja od 2 elementa koja sortiramo insertion sortom)
- $h_2 = 12$
- $h_3 = 6$
- $h_4 = 3$
- $h_5 = 1$ – ovo je klasični insertion sort, ali na polju koje nema ekstrema pa je efikasnost velika

Odabir sekvence razmaka

- Izvor: wikipedia

General term ($k \geq 1$)	Concrete gaps	Worst-case time complexity	Author and year of publication
$\lfloor N/2^k \rfloor$	$\left\lfloor \frac{N}{2} \right\rfloor, \left\lfloor \frac{N}{4} \right\rfloor, \ldots, 1$	$\Theta(N^2)$ [when $N=2^p$]	Shell, 1959[1]
$2\lfloor N/2^{k+1} \rfloor + 1$	$2\left\lfloor \frac{N}{4} \right\rfloor + 1, \ldots, 3, 1$	$\Theta(N^{3/2})$	Frank & Lazarus, 1960[3]
$2^k - 1$	$1, 3, 7, 15, 31, 63, \ldots$	$\Theta(N^{3/2})$	Hibbard, 1963[4]
$2^k + 1$, with 1 prepended	$1, 3, 5, 9, 17, 33, 65, \ldots$	$\Theta(N^{3/2})$	Papernov & Stasevich, 1965[5]
successive numbers of the form $2^p 3^q$	$1, 2, 3, 4, 6, 8, 9, 12, \ldots$	$\Theta(N \log^2 N)$	Pratt, 1971[6]
$(3^k - 1)/2$, not greater than $\lceil N/3 \rceil$	$1, 4, 13, 40, 121, \ldots$	$\Theta(N^{3/2})$	Knuth, 1973[7]
$\prod_{\substack{0 \leq q < r \\ q \neq (r^d+r)/2-k}} a_q$, where $r = \left\lfloor \sqrt{2k + \sqrt{2k}} \right\rfloor$, $a_q = \min\{n \in \mathbb{N} : n \geq (5/2)^{q+1}, \forall p : 0 \leq p < q \Rightarrow \gcd(a_p, n) = 1\}$	$1, 3, 7, 21, 48, 112, \ldots$	$O(Ne^{\sqrt{8 \ln(5/2) \ln N}})$	Incerpi & Sedgewick, 1985[8]
$4^k + 3 \cdot 2^{k-1} + 1$, with 1 prepended	$1, 8, 23, 77, 281, \ldots$	$O(N^{4/3})$	Sedgewick, 1986[9]
$9(4^{k-1} - 2^{k-1}) + 1, 4^{k+1} - 6 \cdot 2^k + 1$	$1, 5, 19, 41, 109, \ldots$	$O(N^{4/3})$	Sedgewick, 1986[9]
$h_k = \max\{\lfloor 5h_{k-1}/11 \rfloor, 1\}, h_0 = N$	$\left\lfloor \frac{5N}{11} \right\rfloor, \left\lfloor \frac{5}{11} \left\lfloor \frac{5N}{11} \right\rfloor \right\rfloor, \ldots, 1$	?	Gonnet & Baeza-Yates, 1991[10]
$\left\lceil \frac{9^k - 4^k}{5 \cdot 4^{k-1}} \right\rceil$	$1, 4, 9, 20, 46, 103, \ldots$	?	Tokuda, 1992[11]
unknown	$1, 4, 10, 23, 57, 132, 301, 701$	?	Ciura, 2001[12]

Shell sort implementacija

- Implementacija shell sorta s originalnom sekvencom razmaka iz 1959:

```
for (int step = n / 2; step > 0; step /= 2) {

   for (int i = step; i < n; i++) {
      int temp = data[i];
      for (int j = i; j >= step && data[j - step] > temp; j -= step) {
         swap(data[j], data[j - step]);
      }
   }

}
```

ALGEBRA BERNAYS SVEUČILIŠTE

31

MERGE SORT

ALGEBRA BERNAYS SVEUČILIŠTE

32

Merge sort

Mergesort Ω(n log(n)) Θ(n log(n)) O(n log(n))

- Temelji se na činjenici: dva već sortirana polja je jednostavno spojiti u jedno veće uz održavanje sortiranosti
  - Algoritam početno dijeli polje na dva (pod)jednaka dijela
  - Svaki od tih dijelova ponovo dijeli na dva dijela sve dok ne dođe do dijela veličine 1
    - Njegovi elementi su po definiciji već sortirani
  - U svakom sljedećem koraku, algoritam spaja dva po dva dijela
  - U zadnjem koraku spaja dva dijela u jedan i time dobijemo sortirano polje
- Izvodi se rekurzijom

ALGEBRA BERNAYS SVEUČILIŠTE

33

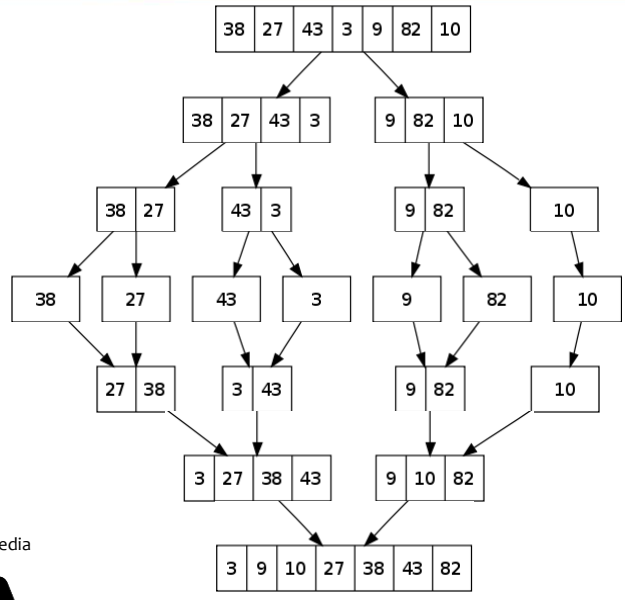
Spajanje dva sortirana polja u jedno

- Osnova merge sorta je efikasno spajanje dva sortirana polja u jedno, uz zadržavanje sortiranosti
- Algoritam je sljedeći:
  - Imamo dva sortirana polja *a* i *b* i jedno prazno polje *c* dovoljne veličine
  - Pratimo tri indeksa: sljedeći u *a*, sljedeći u *b* i sljedeći u *c*
  - Uspoređujemo vrijednosti na sljedećim indeksima u *a* i *b* i manju vrijednost upisujemo u *c*
    - Povećavamo sljedeći indeks u *a* ili *b*, te uvijek u *c*
  - Ponavljamo do kraja (ako u međuvremenu iscrpimo jedno polje, ostatak drugog samo prepišemo)

ALGEBRA BERNAYS SVEUČILIŠTE

34

Merge sort primjer



Izvor: wikipedia

Merge sort implementacija (1/2)

```
void merge_sort(int data[], int from, int to) {
	if (from == to) { // Uvjet zaustavljanja.
		return;
	}

	int mid = (from + to) / 2;
	merge_sort(data, from, mid);
	merge_sort(data, mid + 1, to);

	merge(data + from, mid - from + 1, data + mid + 1,
	  to - mid);
}
```

Merge sort implementacija (2/2)

```
void merge(int poljea[], int na, int poljeb[], int nb) {
     int* poljec = new int[na + nb];
     int ia = 0, ib = 0;
     for (int ic = 0; ic < na + nb; ic++) {
          if (ia == na) { // Ispraznili smo polje a.
               poljec[ic] = poljeb[ib++];
               continue;
          }
          if (ib == nb) { // Ispraznili smo polje b.
               poljec[ic] = poljea[ia++];
               continue;
          }
          if (poljea[ia] < poljeb[ib]) {
               poljec[ic] = poljea[ia++];
          }
          else {
               poljec[ic] = poljeb[ib++];
          }
     }
     for (int i = 0; i < na + nb; i++) {
          poljea[i] = poljec[i];
     }
     delete[] poljec;
}
```


Dodatni primjer

- Pogledati primjer na cs.usfca.edu/~galles/visualization/ComparisonSort.html

QUICK SORT

39

Quick sort

Quicksort Ω(n log(n)) θ(n log(n)) O(n^2)

- Efikasan algoritam, troši malo resursa
- Taktika:
  - Odredi pivotni element
  - Elemente manje od njega prebaci lijevo, veće prebaci desno. Sad je pivotni element na finalnom mjestu, te imamo dio s manjim elementima i dio s većim elementima
    - Jednake možemo staviti na jednu ili drugu stranu
  - Rekurzivno napravi prethodne korake na dijelu s manjim i na dijelu s većim elementima
- Rekurzivan algoritam
  - ...e koja ima 0 ili 1 element ne treba sortirati (bazni slučaj)

40

Detaljniji opis (1/2)

- Detaljniji opis quick sorta:
  - Prvi element u dijelu koji sortiramo proglasimo pivotom
  - Krećemo od drugog elementa i idemo u desno tražeći prvi element veći od pivota (`i`)
  - Krećemo od zadnjeg elementa i idemo u lijevo tražeći prvi element manji ili jednak pivotu (`j`)
  - Ako je `i` < `j`, zamijenimo te elemente pa opet `i` ide u desno, `j` u lijevo
  - Kad nam `i` prestigne `j`, zamijenimo pivot sa `j`
    - Pivot je sad na svom mjestu
    - Svi elementi lijevo su manji ili jednaki pivotu
    - Svi elementi desno su veći od pivota

ALGEBRA BERNAYS SVEUČILIŠTE

41

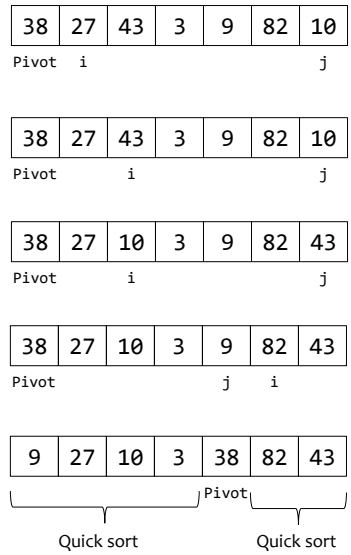
Detaljniji opis (2/2)

- Sad formiramo dva polja:
  - Prvo polje čine svi elementi lijevo od pivota
  - Drugo polje čine svi elementi desno od pivota
- Na svakom polju rekurzivno napravimo quick sort
  - Stanemo kad dobijemo polje veličine 0 ili 1 jer je to polje sigurno sortirano

ALGEBRA BERNAYS SVEUČILIŠTE

42

Quick sort primjer



43

Dodatni primjer

- Pogledati primjer na cs.usfca.edu/~galles/visualization/ComparisonSort.html

44

Quick sort implementacija

```
void quick_sort(int* data, int left, int right) {
        if (right <= left) {
                return;
        }
        int& pivot = data[left];
        int i = left + 1;
        int j = right;
        while (i <= j && i <= right && j > left) {
                while (data[i] <= pivot && i <= right) { // Pomičemo i u desno.
                        i++;
                }
                while (data[j] > pivot && j > left) { // Pomičemo j u lijevo.
                        j--;
                }
                if (i < j) { // Zamijenimo ih.
                        swap(data[i], data[j]);
                }
        }
        swap(pivot, data[j]);
        quick_sort(data, left, j - 1);
        quick_sort(data, j + 1, right);
```

ALGEBRA BERNAYS SVEUČILIŠTE

45

Odabir pivotnog elementa

- Odabir pivotnog elementa može značajno poboljšati, ali i pokvariti efikasnost quick sort algoritma
  - U prvim (i našim) verzijama se uzimao krajnje lijevi element
    - Problem: ako je polje već sortirano, imamo najgori slučaj
- Postoji nekoliko varijanti izbora pivota:
  - Uzeti slučajni element
  - Uzeti srednji element
  - Uzeti medijan (vrijednost u sredini) prvog, srednjeg i zadnjeg elementa
    - Bitno se smanjuje vjerojatnost za ostvarivanje najgoreg slučaja
      - Morala bi sva tri elementa biti među najvećima ili najmanjima u polju

ALGEBRA BERNAYS SVEUČILIŠTE

46

HEAP SORT

47

Heap sort

Heapsort	Ω(n log(n))	θ(n log(n))	O(n log(n))

- Vrlo jednostavna implementacija pomoću STL-a:

```
make_heap(data, data + n);
sort_heap(data, data + n);
```

48

Zadatak

1. Promijenite selection sort tako da sortira pravokutnike (širina, visina) padajuće prema površini. Učitajte svih 1000 pravokutnika iz pravokutnici.txt (u svakom retku su širina i visina jednog pravokutnika odvojeni razmakom) i ispišite ih sortirano rastuće.

ALGEBRA BERNAYS SVEUČILIŠTE

49

Rješenje – funkcija za sortiranje

```
void selection_sort(pravokutnik data[], int n) {
  for (int i = 0; i < n - 1; i++) {
    int min_index = i;
    for (int j = i + 1; j < n; j++) {
      if (data[j].povrsina() > data[min_index].povrsina()) {
        min_index = j;
      }
    }
    swap(data[min_index], data[i]);
  }
}
```

ALGEBRA BERNAYS SVEUČILIŠTE

50

Rješenje – main

```
ifstream dat("pravokutnici.txt");
const int BROJ_ELEMENATA = 1000;

pravokutnik* pravokutnici = new pravokutnik[BROJ_ELEMENATA];

for (int i = 0; i < BROJ_ELEMENATA; i++) {
    dat >> pravokutnici[i].a;
    dat >> pravokutnici[i].b;
}
dat.close();

selection_sort(pravokutnici, BROJ_ELEMENATA);
for (int i = 0; i < BROJ_ELEMENATA; i++) {
    cout << pravokutnici[i].a << " " << pravokutnici[i].b
        << " (" << pravokutnici[i].povrsina() << ")" << endl;
}

delete[] pravokutnici;
```

ALGEBRA BERNAYS SVEUČILIŠTE

51