Sortiranje pomoću hrpe (heapsort)

- Na prijašnjim vježbama već smo radili dva algoritma sortiranja (sortiranje umetanjem - Insertion Sort i mjehuričasto sortiranje - Bubble Sort) koji su ulazni niz od n elemenata sortirali u vremenu proporcionalnom s n^2.
- Sad ćemo obraditi još jedan algoritam sortiranja – sortiranje pomoću hrpe - Heapsort – koji je efikasniji od Insertion Sorta i Bubble Sorta. Ovaj algoritam polje od *n* elemenata sortira u vremenu O(n*log n).
- Prije nego izložimo algoritam za ovo sortiranje, slijedi najosnovnije o stablastoj strukturi podataka zvanoj **hrpa**.
- Potpuno binarno stablo T je hrpa (heap) ako su ispunjeni uvjeti:

  - čvorovi od T su označeni podacima nekog tipa za koje je definiran totalni uređaj

  - neka je *i* bilo koji čvor od T. Tada je oznaka od *i* manja ili jednaka od oznake bilo kojeg djeteta od *i* – *minimalna hrpa*
- također, može biti ovako:

  - neka je *i* bilo koji čvor od T. Tada je oznaka od *i* veća ili jednaka od oznake bilo kojeg djeteta od *i* –*maksimalna hrpa*

Općenito: uređaj među podacima može biti i neki drugi, pa se može promijeniti i relacija roditelj - djeca

- uzimamo da je oznaka roditelja veća ili jednaka od oznaka svih potomaka
- Hrpa je struktura podataka koja se obično implementira pomoću polja
- elementi hrpe spremljeni su u polje koje promatramo kao potpuno binarno stablo (jer je svaki nivo stabla, osim posljednjega, do kraja ispunjen, a čvorovi na posljednjem nivou su "gurnuti" u lijevu stranu)
- Lako je vidjeti da se lijevo dijete čvora i nalazi na poziciji $2*i+1$, a desno na poziciji $2*i+2$. Roditelj čvora i nalazi se na poziciji $(i-1)/2$
- Za svaki čvor u stablu, definiramo njegovu **visinu** kao broj grana na najdužem putu od tog čvora do nekog lista. **Visinu stabla** definiramo kao visinu njegovog korijena
- Budući da je n-elementna hrpa poseban slučaj potpunog binarnog stabla, njezina visina je jednaka $\log_2 n$. Osnovna operacija za rad s hrpom (Heapify) svoj posao obavlja u vremenu koje je, u najgorem slučaju, proporcionalno visini stabla i stoga se izvršava u vremenu proporcionalnom $\log_2 n$
- Operacije za rad s hrpom su sljedeće:
  - Funkcija Heapify koja je ključna za očuvanje uređenosti hrpe ($\log_2 n$)
  - Funkcija BuildHeap koja od neuređenog ulaznog polja stvara hrpu ($n*\log_2 n$)
  - Funkcija HeapSort koja sortira polje ($n*\log_2 n$)

Implementacija strukture podataka HeapType u C-u

- definiramo novi tip podataka, HeapType, kao strukturu koja sadrži:

```
struct heaptype {
      int * Elements;          //pokazivač na polje elemenata hrpe
      int Size;                //veličina hrpe
 };
typedef struct heaptype HeapType;
```

- Funkcija MakeEmptyHeap stvara praznu hrpu zadane veličine, dok funkcija FreeHeap oslobodi memoriju koju je neka hrpa zauzimala:

```
HeapType MakeEmptyHeap(int size) {
   HeapType heap;
   heap.Elements = malloc(size * sizeof(int));
   heap.Size = size;
   return heap;
 }

 void FreeHeap(HeapType * hptr) {
    free(hptr->Elements);
   heapptr->Size=0;
 }
```

Operacija Heapify

- Heapify je operacija koja kao ulazni parametar prima pokazivač na strukturu HeapType i index *i* nekog elementa u polju Elements. U času pozivanja funkcije Heapify pretpostavlja se da i lijevo i desno podstablo čvora *i* zadovoljavaju svojstvo uređenosti hrpe, ali da je element s indeksom *i* (eventualno) manji od svoje djece i time (eventualno) narušava uređenost hrpe
- Funkcija Heapify "pomiče" element upisan u čvor *i* prema dolje tako da na kraju podstablo hrpe kojemu je korijen čvor *i* postane ispravna podhrpa (tj. podhrpa koja zadovoljava svojstvo uređenosti hrpe)
- Left i Right su jednostavne pomoćne funkcije koje vraćaju indeks lijevog odnosno desnog djeteta čvora *i*:

```
int Left(int i) {
    return 2*i + 1;
}
```

```
int Right(int i) {
    return 2*i+2;
}
```

```
void Heapify(HeapType * hptr, int i) {
    int largest;
    int lft;
    int rht;
    int next = 1;
    do    {
          lft = Left(i);
          rht = Right(i);
          if(lft < hptr->Size && hptr->Elements[lft] > hptr->Elements[i])
                largest = lft;
          else
                largest = i;
          if(rht < hptr->Size && hptr->Elements[rht] > hptr->Elements[largest])
                largest = rht;
          if(largest != i)      {
                int temp = hptr->Elements[i];
                hptr->Elements[i] = hptr->Elements[largest];
                hptr->Elements[largest] = temp;
                i = largest;
          }
          else
                next = 0;
    }
    while(next);
 }
```

Operacija BuldHeap

- Sukcesivnom upotrebom operacije Heapify možemo od "neispravne" hrpe načiniti ispravnu (koja zadovoljava uređenost hrpe). Funkcija izgleda ovako:

```
void BuildHeap(HeapType * heapptr) {
    for (int i = ParentOfLastElement(heapptr); i >= 0; i--)
        Heapify(heapptr, i);
}
```

- BuildHeap koristi pomoćnu funkciju ParentOfLastElement koja vraća indeks roditelja posljednjeg elementa u hrpi. Ta funkcija izgleda ovako:

```
int ParentOfLastElement(HeapType * h) {
  return Parent(h->Size - 1);
 }

 int Parent(int k) {
    return (k - 1) / 2;
  }
```

Algoritam HeapSort

- Algoritam Heapsort najprije pomoću funkcije BuildHeap od ulaznog polja stvori ispravnu hrpu. Nakon toga, stvorenu hrpu pretvara u uzlazno sortirano polje. Funkcija izgleda ovako:

```
void HeapSort(int a[], int n) {
    HeapType heap;
    heap.Elements = a;
    heap.Size = n;
    BuildHeap(&heap);
    for(int i = n - 1; i > 0; i--)
    {
            int temp = heap.Elements[i];
            heap.Elements[i] = heap.Elements[0];
            heap.Elements[0] = temp;
            heap.Size--;
            Heapify(&heap, 0);
    }
}
```

- Vidimo da algoritam najprije poziva BuildHeap (što traje proporcionalno s $n*\log_2 n$) a zatim u osnovi "vrti petlju" u kojoj se n-1 puta obavljaju neke jednostavne operacije i poziva funkcija Heapify koja traje proporcionalno s $\log_2 n$. Sveukupno, stoga, funkcija HeapSort se izvršava u vremenu proporcionalnim s $n*\log_2 n$, što je bitno bolje od sortiranja umetanjem (Insertion Sort) ili mjehuričastog sortiranja (Buble Sort) čije je vrijeme izvršavanja dano s n^2 .

- Kompletan kod za sortiranje pomoću hrpe: primjer

- Napisati program koji od ulaznog polja cijelih brojeva izgradi hrpu, te ispiše polje u kojem su spremljeni elementi hrpe. Također napraviti sortiranje polja cijelih brojeva upotrebom sortiranja pomoću hrpe.

```
#include <stdio.h>
#include <stdlib.h>
#define MaxSize 20

struct heaptype {
      int * Elements;
      int Size;
};

typedef struct heaptype HeapType;

int Left(int k) {
      return 2*k + 1;
}

int Right(int k) {
   return 2*k + 2;
}

int Parent(int k) {
      return (k - 1) / 2;
}

int ParentOfLastElement(HeapType * h) {
      return Parent(h->Size - 1);
}
```

```
HeapType MakeEmptyHeap(int size) {
    HeapType heap;
    heap.Elements = malloc(size * sizeof(int));
    heap.Size = size;
    return heap;
}

void FreeHeap(HeapType * heapptr) {
    free(heapptr->Elements);
    heapptr->Size=0;}

void Heapify(HeapType * heapptr, int i) {
    int largest, lft, rht, dalje = 1;
    do    {
          lft = Left(i);          rht = Right(i);
          if(lft < heapptr->Size && heapptr->Elements[lft] > heapptr->Elements[i])
                largest = lft;
          else       largest = i;
          if(rht < heapptr->Size && heapptr->Elements[rht] > heapptr->Elements[largest])
                largest = rht;
          if(largest != i)       {
                int temp = heapptr->Elements[i];
                heapptr->Elements[i] = heapptr->Elements[largest];
                heapptr->Elements[largest] = temp;
                i = largest;          }          else
                dalje = 0;}          while(dalje);
}
```

```
void BuildHeap(HeapType * heapptr)
{
    for(int i = ParentOfLastElement(heapptr); i >= 0; i--)
        Heapify(heapptr, i);
}

void HeapSort(int a[], int n)
{
    HeapType heap;
    heap.Elements = a;
    heap.Size = n;
    BuildHeap(&heap);
    for(int i = n - 1; i > 0; i--)
    {  // u korijenu je najveći element hrpe
        int temp = heap.Elements[i];
        heap.Elements[i] = heap.Elements[0];  // korijen stavljamo na kraj polja
        heap.Elements[0] = temp;  // zadnji element polja ide u korijen
        heap.Size--;  // smanjimo hrpu za 1
        Heapify(&heap, 0);  // slozi se nova hrpa s n-1 čvorova
    }
}
```

```
int main() {
    int i, a[MaxSize] = {4,0,11,1,21,-3,26,2,5,16,17,9,10,13,14,25,8,3,7,-9};
    HeapType myheap = MakeEmptyHeap(MaxSize);

    for(i = 0; i < myheap.Size; i++)
        myheap.Elements[i] = a[i];
    printf("Ulazno polje:\n");
    for(i = 0; i < myheap.Size; i++)
        printf("%i ", myheap.Elements[i]);      printf("\n\n");
    BuildHeap(&myheap);
    printf("Nakon kreiranja hrpe:\n");
    for(i = 0; i < myheap.Size; i++)
        printf("%i ", myheap.Elements[i]);      printf("\n\n\n");

    FreeHeap(&myheap);
    printf("Prije sortiranja:\n");
    int b[MaxSize] = {4,0,11,1,21,-3,26,2,5,16,17,9,10,13,14,25,8,3,7,-9};
    for(i = 0; i < MaxSize; i++)
        printf("%i ", b[i]);   printf("\n\n");
    HeapSort(b, MaxSize);
    printf("Nakon sortiranja:\n");
    for(i = 0; i < MaxSize; i++)
        printf("%i ", b[i]);   printf("\n");
    system("PAUSE");
    return 0;
}
```

Drugi primjer izvedbe hrpe

- Jednostavan programski kod koji od elemenata polja stvori hrpu, koristi se jednostavnija struktura podataka (samo polje)
- Ovdje se pretpostavlja da je prvi element hrpe u ćeliji polja indeksa 1 (prva ćelija polja indeksa 0 se ne koristi), pa je roditelj *i*-tog čvora na mjestu *i*//2
- Na "dno" (list) hrpe dodaje se član koji se onda uspoređuje i zamjenjuje ako je potrebno sa svojim roditeljem, praroditeljem, prapraroditeljem itd. dok ne postane manji ili jednak nekoj od tih vrijednosti.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>  //za funkciju pow()
#define MAXGOM 100
typedef int tip;

void ubaci (tip A[], int k) {  // ubacuje vrijednost iz A[k] na hrpu pohranjenu u A[1:k-1]
  int i, j;
  tip novi;
  j = k;
  i = k/2;
  novi = A[k];
  while ((i > 0) && (A[i] < novi)) {
    A[j] = A[i]; // povecaj razinu za roditelja
    j = i;
    i /= 2;    // roditelj od A[i] je na A[i/2]
  }
  A[j] = novi;
}

void main(void) {
  FILE *fi;
  int i, j, k;
  tip A[MAXGOM];
```

```
fi = fopen ("UlazZaHrpu.txt", "r");
  if (fi) {
            j = 1;
            while (j < MAXGOM && fscanf(fi, "%d", &A[j]) != EOF) {
                        printf ("%d. ulazni podatak je %d\n", j, A [j]);
                        ubaci (A, j);
                        j++;
            }
            fclose (fi);
            // ispisi hrpu po retcima
            i = 1;
            k = 1;
            while (i < j) { // petlja do zadnjeg u hrpi
                        // pisi do maksimalnog u hrpi razine k
                        for (; i <= pow (2, k) - 1 && i < j; i++) {
                                    printf(" %d ", A[i]);
                        }
                        k++; // povecaj razinu
                        printf ("\n");
            }
     } else {
            printf ("Nema ulazne datoteke\n");
     }
     system("PAUSE");
     exit(0);
}
```

- Za analizu najgoreg slučaja algoritma uzmimo n elemenata. Na i-toj razini potpunog binarnog stabla ima najviše 2^{i-1} čvorova. Na svim nižim razinama do tada ima ukupno $2^{i-1} - 1$ čvorova, za $i > 1$. Stablo s k razina ima najviše 2^k-1 čvorova. Stablo s k-1 razinom ima najviše $2^{k-1}-1$ čvorova. Ako je stablo potpuno, započeta je posljednja razina, pa vrijedi $2^{k-1} - 1 < n \leq 2^k - 1$
- Iz ovoga slijedi:
  - $2^{k-1} < n + 1 \Rightarrow (k - 1) \log 2 < \log (n + 1) \Rightarrow k < \log_2 (n + 1) + 1$

  $n + 1 \leq 2^k \Rightarrow \log (n+1) \leq k \log 2 \Rightarrow \log_2 (n+1) \leq k$

  $\log_2 (n+1) \leq k < \log_2 (n+1) + 1$ odnosno $k = \lceil log_2(n+1) \rceil$
- Na primjer:
  - za n = 14 treba $\lceil \log_2 15 \rceil = \lceil \ln 15/\ln 2 \rceil = \lceil 2.70805/0.693147 \rceil = \lceil 3.9 \rceil$ = 4 razine
  - za n = 15 treba $\lceil \log_2 16 \rceil = \lceil 4 \rceil$ = 4 razine
  - za n = 16 treba $\lceil \log_2 17 \rceil = \lceil 4.087 \rceil$ = 5 razina

- U najgorem slučaju, `while` petlja se izvršava proporcionalno broju razina u gomili. Skup podataka koji predstavlja najgori slučaj za ovaj algoritam je polje s rastućim podacima. Tada svaki novi element, onaj koji se ubacuje u gomilu pozivom funkcije `ubaci`, postaje korijen pa se kroz k razina obavlja zamjena. Vrijeme izvođenja je tada $O(n \log_2 n)$. Za prosječne podatke vrijeme za stvaranje gomile iz skupa podataka je $O(n)$, što je za red veličine bolje.

Još jedan primjer izvedbe hrpe

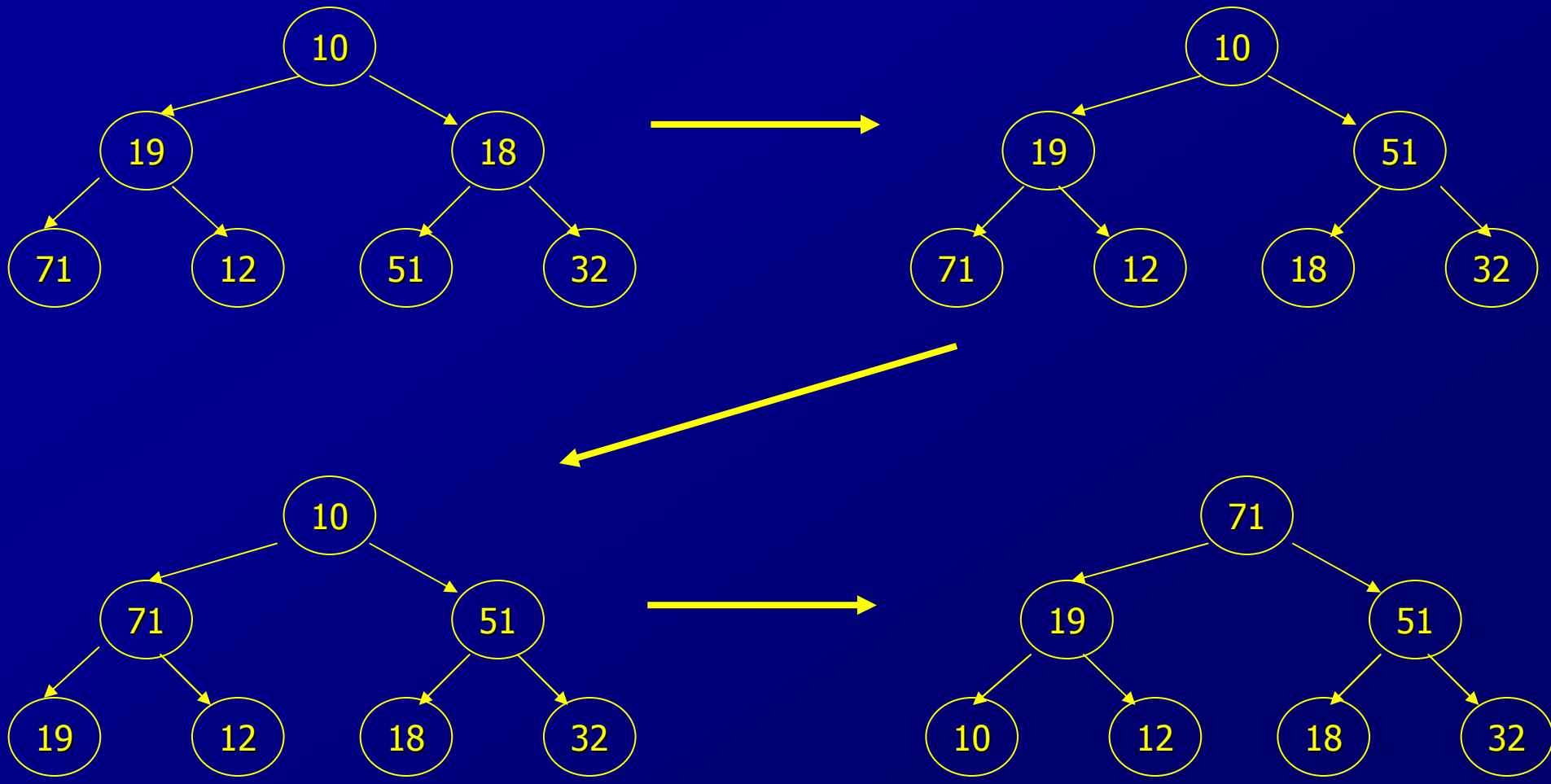
- Za poboljšanje brzine obavljanja zadanih operacija stvoren je algoritam koji kreće od krajnjih čvorova prema korijenu, razinu po razinu. Samo podatak u korijenu može narušavati svojstvo hrpe, dok podstabla zadržavaju to svojstvo. Tada je samo potrebno tu nepravilnost ispraviti i opet dobivamo željenu hrpu. To čini funkcija `podesi` u slijedećem primjeru. Za krajnje čvorove svojstvo hrpe je zadovoljeno, pa treba u `stvori_hrpu` funkciji provesti popravljanje svojstva hrpe samo za korijen stabla (identični postupak prvom primjeru za hrpu)
- Funkcija podesi(): potpuna binarna stabla s korijenima A[2*i] i A[2*i+1]kombiniraju se s A[i] formirajuci jedinstvenu hrpu

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define MAXGOM 100
typedef int tip;
// potpuna binarna stabla s korijenima A[2*i] i A[2*i+1] kombiniraju se s
// A[i] formirajuci jedinstvenu hrpu, 1 <= i <= n
void podesi (tip A[], int i, int n) {
  int j;
  tip stavka;
  j = 2*i;
  stavka = A[i];
  while (j <= n ) {  // Usporedi lijevo i desno dijete (ako ga ima)
    if ((j < n) && (A[j] < A[j+1])) j++;    // j pokazuje na vece dijete
    if (stavka >= A[j]) break;   // stavka je na dobrom mjestu
    A[j/2] = A[j];                  // vece dijete podigni za razinu
    j *=2;  }
  A[j/2] = stavka;
}

// premjesti elemente A[1:n] da tvore gomilu
void StvoriHrpu (tip A[], int n) {
  int i;
  for (i = n/2; i >= 1; i--)
            podesi (A, i, n);
}
```

```
void main(void) {
	FILE *fi;
	int i, j, k, n;
	tip A[MAXGOM];
	fi = fopen ("UlazZaHrpu.txt", "r");
	if (fi) {
		j = 1;
		while (j < MAXGOM && fscanf (fi, "%d", &A[j]) != EOF) {
			printf ("%d. ulazni podatak je %d \n", j, A[j]);
			j++; }
		fclose (fi);
		// podesi broj elemenata i stvori hrpu
		n = j - 1;
		StvoriHrpu (A, n);
		// pisi hrpu po retcima
		i = 1;  k = 1;
		while (i < j) { // petlja do zadnjeg u hrpi
			// pisi do maksimalnog u hrpi razine k
			for (; i <= pow (2, k) - 1 && i < j; i++) {
				printf(" %d ", A[i]);
			}k++; // povecaj razinu
			printf ("\n"); }
	} else {
		printf ("Nema ulazne datoteke\n"); }
	system("PAUSE"); exit(0); }
```

- Stvaranje hrpe za ulazni niz podataka: 10,19,18,71,12,51,32



- Za n podataka, $2^{k-1} \leq n < 2^k$, broj razina je $k = \lceil log_2(n+1) \rceil$. Za najgori slučaj broj iteracija u `podesi` iznosi k-i za čvor na razini i gdje ima najviše 2^{i-1} čvorova. Slijedi da je, vrijeme izvođenja za `StvoriHrpu`:

  $\sum_{i=1}^{k} 2^{i-1} (k-i)$. Uočimo da se eksponent mijenja od 0 do $k-1$, a faktor od $k-1$ do 0.

  Slijedi ekvivalentni izraz kad se izbaci faktor 0 i obrne redosljed sumacije:

  $$= \sum_{i=1}^{k-1} i\, 2^{k-i-1} = \sum_{i=1}^{k-1} 2^{k-1}\, i\, 2^{-i}$$

  S obzirom da je $2^{k-1} \leq n$,

  $$\leq n \sum_{i=1}^{k-1} i/2^i \leq 2n = O(n),$$ jer suma reda teži prema 2.

- Vrijeme izvođenja za najgori slučaj algoritma `StvoriHrpu` je $O(n)$, što je za red veličine bolje od $O(n \log_2 n)$ za uzastopno korištenje `ubaci` funkcije.
- Funkcija `StvoriHrpu` traži da su svi elementi za stvaranje hrpe već prisutni, dok ubaci može ubaciti novi element u hrpu bilo kada. Funkcije koje hrpa treba brzo obaviti i radi kojih je napravljena ta struktura podataka su ubacivanje novih i brisanje najvećeg elementa iz skupa podataka. Brisanje najvećeg podatka se obavlja izbacivanjem korijena i pozivanjem funkcije `podesi`, a ubacivanje novih se radi funkcijom `ubaci`. Tako se postiže da se obje željene funkcije obavljaju u $O(\log_2 n)$ vremenu.

Prioritetni red i hrpa

- Napisati program za ubacivanje i skidanje elemenata (prioritet određuje cijeli broj) iz prioritetnog reda realiziranog pomoću hrpe. Za prioritet elementa i odabir operacije (ubacivanje ili skidanje) koristi se generator slučajnih brojeva. Ako generator načini parni broj, element najvećeg prioriteta vadi se iz prioritetnog reda. Ako generator načini neparni broj, stvara se novi element sa slučajno generiranim prioritetom i stavlja u prioritetni red.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <ctype.h>
#define MAXPRIOR 100
typedef int tip;

void podesi (tip A[], int i, int n) {
     // potpuna binarna stabla s korijenima A[2*i] i A[2*i+1]
     // kombiniraju se s A[i] stvarajuci jedinstvenu hrpu, 1 <= i <= n
     int j;
     tip stavka;
     j = 2 * i;
     stavka = A[i];
     while (j <= n) {
          // Usporedi lijevo i desno dijete (ako ga ima)
          if ((j < n) && (A[j] < A[j + 1])) j++;
          // j pokazuje na vece dijete
          if (stavka >= A[j]) break;       // stavka je na dobrom mjestu
          A[j / 2] = A[j]; // vece dijete podigni za razinu
          j *= 2;
     }
     A[j / 2] = stavka;
}
```

```
void ubaci (tip A[], int k) {   // ubacuje vrijednost iz A[k] na hrpu pohranjenu u A[1 : k - 1]
    int i, j;
    tip novi;
    j = k;
    i = k / 2;
    novi = A[k];
    while ((i > 0) && (A[i] < novi)) {
        A[j] = A[i]; // smanji razinu za roditelja
        j = i;
        i /= 2;        // roditelj od A[i] je na A[i/2]
    }
    A[j] = novi;
}

tip skini (tip A[], int *k) {
    // izbacuje vrijednost iz A[k] sa prvog mjesta, ako je red prazan vraca -1
    tip retVal = -1;
    if (*k <= 1) return retVal;
    retVal = A[1];
    (*k) --;
    A[1] = A[*k];
    podesi (A, 1, *k);
    return retVal;
}
```

```
int main() {
    int prior, i, j, k = 1;
    tip A[MAXPRIOR];
    srand((unsigned) time(NULL));
     printf("Za obavljanje jednog koraka pritisni ENTER, za kraj bilo koji znak\n”);
    while(isspace(getchar())) {
        if (rand() % 2) {
            if (k >= MAXPRIOR)
                printf(“Prioritetni red je pun!\n");
            else {
     printf("Dodavanje u prioritetni red: %d\n", prior=(int)(rand()/(RAND_MAX + 1.) * 99 + 1));
                A[k] = prior;
                ubaci(A, k);
                k++; }
        } else {
            if ((prior = skini(A, &k)) == -1)
                printf(“Prioritetni red je prazan!\n");
            else
                printf("Skidanje iz prioritetnog reda: %d\n", prior); }
        for (i = 1, j = 1; i < k; j++) {
            for (; i <= pow (2, j) - 1 && i < k; i++) {
                printf(" %d ", A[i]); }
            printf ("\n"); } }
  system(“PAUSE”);
}
```