ASD, egzamin – 27.06.2017

1. (15p) Niech $f \prec g$ oznacza, że $f \in O(g)$ i $g \notin O(f)$ a $f \approx g$ oznacza, że $f \in O(g)$ i $g \in O(f)$. Niech c, d będą stałymi takimi, że $4 < c < 5 < d < 6$. Uporządkuj następujące funkcje: $d * n^3 * (\log_2(n))^2 + c * n^2 * (\log_2(n))^3$, $(c+1) * n^3 * (\log_2(n))^2$, $2^{c*\log_2(n)}$, $n^{\log_2(n)}$.

2. (15p) Wyznacz f taką, że czas działania algorytmu znajduje się w $\Theta(f)$:
```
Hehe(int *t, int poczatek, int koniec){
  int suma:=0; int skok = (koniec - poczatek)/2;
  if(skok <= 10) return 1;
  for(i:=poczatek to i=koniec) suma:=suma+t[i];
  suma:=suma - Hehe(t,poczatek, poczatek+skok);
  for(j:=poczatek +skok to j=koniec) suma:=suma + t[j];
  if(t[poczatek]<t[koniec]) suma:= suma - Hehe(t,poczatek+skok,koniec);
  else suma:=suma+ Hehe(t,poczatek+skok,koniec);
  return suma; }%Hehe
```

3. (15p) Wypisz drzewo rekurencyjnych wywołań dla algorytmu quicksort(t,0,7), dla $t[0 \ldots 7] = \{7, 3, 6, 9, 11, 1, 18, 5\}$. Wypisz aktualną zawartość tablicy t przy każdym wywołaniu quicksort. Przyjmij następującą wersję algorytmu quicksort.

```
int partition(int t, int lewy, int prawy){
 int v=t[prawy];
 int i=lewy-1, j=prawy;
 if(lewy>=prawy)
   return prawy;
 while(i<j){
   do i++;while(t[i]<v && i<j);
   do j-;while(t[j]>v && i<j);
   if(i<j)
     swap(t,i,j);
 }
 swap(t,prawy,i);
 return i;
}

 void quicksort(int *t, int lewy, int prawy){
 if(lewy>=prawy) return;
 int i=partition(t,lewy,prawy);
 quicksort(t,lewy, i-1);
 quicksort(t,i+1,prawy);
}
```

4. (15p) Niech i(x) oznacza instrukcję wstawiania elementu do drzewa BST, d(x) – usuwania. Narysuj drzewo BST, po wykonywaniu ciągu instrukcji: i(10), i(6), i(15), i(12), i(14), i(13), i(15), i(20) (5pt). Narysuj drzewo BST powstałe po wykonaniu instrukcji d(10) (przyjmij, że wybierasz podczas usuwania najmniejszy element z prawego podrzewa) (5pt). Rotacjami w lewo i w prawo wypromuj element 13 do korzenia, narysuj kolejne drzewa BST powstałe po przesunięciach (10pt).

5. (15p) Drzewo jest lianą jeśli każdy jego element jest albo liściem albo ma dokładnie jedno dziecko. Napisz funkcję `int liana(node *root)`, która zwróci 1 jeśli drzewo na które wskazuje `root` jest lianą i 0 w przeciwnym przypadku. Przyjmij, że `node` jest strukturą postaci `struct node{int key; node *left,*right;}`. Puste drzewo oczywiście jest lianą.

6. (15p) Prawdopodobieństwa wyszukiwania elementu o danym kluczu są podane w nawiasach: a(0,2), b(0), c(0,2), d(0,2), e(0,3), f(0,1). W strukturze słownikowej trzymamy elementy a, b, c, e. Oblicz średni koszt wyszukania elementu na optymalnej liście nieuporządkowanej(5pt) i w optymalnym drzewie BST (uzasadnij wybór drzewa BST) (10pt). Rozstrzygnij, która z list będzie efektywniejszą implementacją słownika dla podanych elementów (10pt).

   Przyjmij jako koszt wyszukania elementu ilość odwiedzonych elementów w strukturze. Pamiętaj o wpływie na średni koszt wyszukiwań elementów, których nie ma w strukturze.