AiSD, egzamin – 30.05.2014

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 $g \in O(f)$. Uporządkuj następujące funkcje: $n(\log_2(n))^3$, $2^{(\log(n))^2}$, $n\sqrt{n}$, $2^{\sqrt{n}}$.

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

3. (15p) Przesortuj tablicę $t[1 \ldots 8] = \{5, 2, 8, 3, 9, 1, 7, 2\}$ algorytmem mergesort. Wypisz drzewo wywołań rekurencyjnych w postaci MS(t, i, j) oraz zawartość tablicy w momencie każdego wywołania (korzeń drzewa to MS(t,1,8), [5,2,8,3,9,1,7,2]).

   Przyjmij, że mergesort kończy wywołania rekurencyjne jeśli tablica ma wielkość 2 lub 1 oraz, że dzieli tablicę $t[k \ldots n]$ na części $[k \ldots (k + m - 1)]$ oraz $[(k + m) \ldots n]$, dla $m = \lfloor (n - k + 1)/2 \rfloor$.

4. (15p) Wypisz kolejne drzewa BST powstałe przez wykonanie następujących operacji na pustym drzewie: i(3), i(8), i(6), i(7), i(5), d(7), d(5). Przyjmij, że i(x) oznacza operację wstawiania do drzewa BST, która wstawia element x do korzenia drzewa. Przyjmij, że usuwając element (operacja d(x)), który nie jest liściem wpisujemy na jego miejsce odpowiedni element z jego prawego podrzewa (jeśli oba podrzewa są niepuste).

   Za przedstawienie operacji i(x) jako wstawiania do liści tylko 5p. Za poprawne usuwanie 5p.

5. (15p) Napisz algorytm `NieLiscie(node *)`, który wywołany dla argumentu `tree` zwróci ilość tych elementów drzewa, które nie są liśćmi. Przyjmij, że `node` jest strukturą postaci `struct node{int key; node *left; node *right;}`. Pamiętaj o przypadku gdy argument funkcji jest wskazaniem pustym (null).

6. (15p) Prawdopodobieństwa wyszukiwania elementu o danym kluczu są podane w nawiasach: a(0.1), b(0.2), c(0.2), d(0.05), e(0.1), f(0.2), g(0.05), h(0.1). Na liście trzymamy tylko elementy a, b, d, f, g,h. Oblicz średni koszt wyszukania elementu (zakończonego powodzeniem lub nie) zdefiniowany jako liczba odwiedzonych na liście elementów dla listy uporządkowanej alfabetycznie oraz dla optymalnej listy nieuporządkowanej. Rozstrzygnij, która z tych struktur będzie efektywniejsza.