ASD, egzamin – 12 września, 2017

1. (15p) Określ porządek asymptotycznego wzrostu $g \prec g$ (lub $f \approx g$) dla następujących funkcji (dla $n \geqslant 1$; c, d stałe takie, że $1 < c < 2 < d < 3$): $2^{((\log_2(n))^3)}$, $2^{d \log_2(n)}$, $dn^2 \log_2(n)$, $n(\log_2(n))^4$.

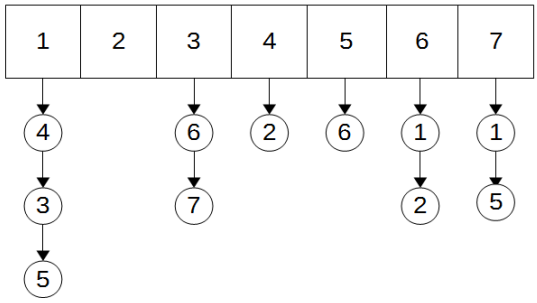
2. (15p) Oszacuj z dokładnością do O(.) czas działania algorytmu (wzgl. rozmiaru tablicy n = koniec - poczatek + 1).

```
Blublu(int *t, int poczatek, int koniec){
  int rozmiar=koniec - poczatek;
  int skok=⌊rozmiar/2⌋;
  if(skok <= 3) return;
  for(i:=0 to i=skok-1)
    print(t[poczatek + i]+t[poczatek+skok+i]);
  if(t[0]>0)
    BluBlu(t,poczatek, poczatek+skok);
  else
    BluBlu(t,poczatek+skok,koniec);
  BluBlu(t,poczatek+1, poczatek+skok+1);
}
```

3. (15p) Przesortuj tablicę $t[1 \ldots N] = \{1, 8, 9, 4, 5, 7\}$ sortowaniem przez kopcowanie. Przyjmij, że kopiec tworzony jest w pętli
   `for(i=2;i<=N;i++) upheap(t,i);`
   Wypisz wartość tablicy po każdym dodaniu elementu do kopca oraz po każdym umieszczeniu elementu na końcowym miejscu i poprawieniu kopca.

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(3), i(8), i(20), i(12), i(30), i(25), i(13), i(15) (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 (5pt).

5. Narysuj graf skierowany reprezentowany przez poniższą strukturę (5p). Przedstaw w jakiej kolejności odwiedzi wierchołki funkcja visit_bfs(3), która przeszukuje graf wszerz (breadth first search). Przyjmij, że początkowo dla każdego wierzchołka x, mark[x] nie jest równe visited (10p), Q jest strukturą reprezentującą kolejkę.




```
visit_bfs(x){
   Q=();
   Q.enqueue(x); mark[x]=visited;
   while(not Q.empty()){
      v=Q.dequeue();
      Edge *ptr=edges[v];
      while(ptr!=NULL){
         if(mark[ptr->end]!=visited){
            Q.enqueue(ptr->end); mark[ptr->end]=visited;
         }
         ptr=ptr->next;
      }
   }
```

6. (15p) Napisz funkcję *int Parzystosc(Node *ptr)*, która zwróci 0 jeśi drzewo ptr ma parzyście wiele wierzchołków i 1 w przeciwnym przypadku (czyli funkcja zwracą liczbę wierzchołków drzewa modulo 2). Przyjmij, że *Node* to struktura postaci *struct Node{int key; Node *left; Node *right; }*.