MCMD L6 : I/O-Graph Algorithms

Disk <---I/O---> RAM <--> CPU
N = size of problem
B = block size
M = size of memory
T = size of output
I/O = block move between disk + memory

Sorting N items:
 sort(N) = Theta((N/B) log_{M/B} (N/B)) << N log_2 N

I/O-Efficient Priority Queue:
 e.g. "Buffer Tree" Arge 96   or CGGTVV
 size O(N/B)
 INSERT = O((1/B) log_{M/N} (N/B)) amortized
 DELETEMIN = O((1/B) log_{M/N} (N/B)) amortized

Like B-Tree, but a size M/B buffer at each internal node.
Only push an element down the tree if the "buffer" is full.

Simplest Graph problem: "List Ranking"
  Given unordered linked list (each link may jump to new block)
  Compute the rank (number of proceeding nodes) from root.
  O(sort(N)) I/Os.

Can then order the linked list in blocks in sort(N) I/Os

(We'll come back to this)

INDEPENDENT SET (of un-ordered linked list)

Make a pass on (un-ordered) nodes:
 Assign node v, p(v) = 0 or 1 at random.
 Put in priority queue PQ at value of pointed to node, with p(v)

Make a pass on (un-ordered) nodes,
  If p(v) = 0 and DELETEMIN has value 1 (of node pointed to v)
    put in I

time = sort(N).

```
E[|I|] >= N/4.
 (possible to find I in sort(N) with |I| > N/3)
-------------

all r(v) = 1
LISTRANKING (L)
  If |L| < M, process in memory
I <- INDEPENDENT SET(L)   (a stack in order of scan)

(create new list L' with L \ I)
scan 1:
 if v in L \ I
   succ_L'(v) <- succ_L(v)
   put v in PQ1 at location succ_L(v)
scan 2:
 if a in I
   v <- DELETEMIN(PQ1)
   put v in PQ2 with location v and store succ_L(a)
   put a in PQ3 with location succ_L(a)
scan 3:
 if v in L \ I
  if DELETEMIN(PQ2)==v  with pointer = s
   set succ_L'(v) = s
  if DELETEMIN(PQ3)==v
   set r(v) += 1   (means one node was skipped over)
  push v to STACK = L'

LISTRANKING(L')
  now all nodes v in L' have correct rank r(v)

scan 4:
  if (succ_L(v) != succ_L'(v))
    put r(v) in PQ4 at succ_L(v)

scan 5:
  if v in I
   DELTEMIN ->  r
   r(v) = r+1

------------

Similar algorithm works for ordering a TREE or DAG from root to leaf.

(recall after this we can sort by r(v))
```

```
-----------------------------------
DAG-HEIGHT:
  "time-forward processing"

assume topologically ordered
  "each v comes after all nodes u with edges (u,v)"

Scan data:
  Root has h(v) = 0.
  Process v:  DELETEMIN while output has value (v,s_i) for some s_i ->
{s_1,...,s_k}.
    Set h(v) = min{s_1,...,s_k} + 1
-----------------------------------
```