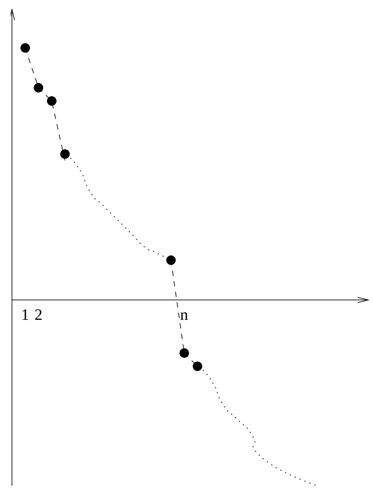
Practice problems: Divide and conquer

1. **(exam1 fall 2003)** In this problem we consider a *monotonously decreasing* function $f : N \rightarrow Z$ (that is, a function defined on the natural numbers taking integer values, such that $f(i) > f(i+1)$). Assuming we can evaluate f at any i in constant time, we want to find $n = \min\{i \in N | f(i) \leq 0\}$ (that is, we want to find the value where f becomes negative).



We can obviously solve the problem in $O(n)$ time by evaluating $f(1), f(2), f(3), \ldots f(n)$. Describe an $O(\log n)$ algorithm. (*Hint:* Evaluate f on $O(\log n)$ carefully chosen values $\leq n$ and possibly at a couple of values between n and $2n$ - but remember that you do not know n initially).

2. **(exam1 fall 2003)** The *maximum partial sum* problem (MPS) is defined as follows. Given an array $A[1..n]$ of integers, find values of i and j with $1 \leq i \leq j \leq n$ such that

$$\sum_{k=i}^{j} A[k]$$

is maximized.

Example: For the array [4,-5,6,7,8,-10,5], the solution to MPS is $i = 3$ and $j = 5$ (sum 21).

To help us design an efficient algorithm for the maximum partial sum problem, we consider the *left position ℓ maximal partial sum* problem ($LMPS_\ell$). This problem consists of finding value j with $\ell \leq j \leq n$ such that

$$\sum_{k=\ell}^{j} A[k]$$

is maximized. Similarly, the *right position r maximal partial sum* problem ($RMPS_r$), consists of finding value i with $1 \leq i \leq r$ such that

$$\sum_{k=i}^{r} A[k]$$

is maximized.

Example: For the array [4,-5,6,7,8,-10,5] the solution to e.g. $LMPS_4$ is $j = 5$ (sum 15) and the solution to $RMPS_7$ is $i = 3$ (sum 16).

(a) Describe $O(n)$ time algorithms for solving $LMPS_\ell$ and $RMPS_r$ for given ℓ and r.

(b) Using an $O(n)$ time algorithm for $LMPS_\ell$, describe a simple $O(n^2)$ algorithm for solving MPS.

(c) Using $O(n)$ time algorithms for $LMPS_\ell$ and $RMPS_r$, describe an $O(n \log n)$ divide-and-conquer algorithm for solving MPS.

3. Suppose you are given an array $A[1..n]$ of sorted integers that has been *circularly shifted* k positions to the right. For example, $[35, 42, 5, 15, 27, 29]$ is a sorted array that has been circularly shifted $k = 2$ positions, while $[27, 29, 35, 42, 5, 15]$ has been shifted $k = 4$ positions.

   We can obviously find the largest element in A in $O(n)$ time. Describe an $O(\log n)$ algorithm.

4. In this problem we consider divide-and-conquer algorithms for building a heap H on n elements given in an array A. Recall that a heap is an (almost) perfectly balanced binary tree where $\text{key}(v) \geq \text{key}(\text{parent}(v))$ for all nodes v. We assume $n = 2^h - 1$ for some constant h, such that H is perfectly balanced (leaf level is "full").

   First consider the following algorithm SlowHeap$(1, n)$ which constructs (a pointer to) H by finding the minimal element x in A, making x the root in H, and recursively constructing the two sub-heaps below x (each of size approximately $\frac{n-1}{2}$).

   SlowHeap(i, j)

   If $i = j$ then return pointer to heap consisting of node containing $A[i]$

   Find $i \leq l \leq j$ such that $x = A[l]$ is the minimum element in $A[i \ldots j]$

   Exchange $A[l]$ and $A[j]$

   $Ptr_{\text{left}} = \text{SlowHeap}(i, \lfloor \frac{i+j-1}{2} \rfloor)$

   $Ptr_{\text{right}} = \text{SlowHeap}(\lfloor \frac{i+j-1}{2} \rfloor + 1, j - 1)$

   Return pointer to heap consisting of root r containing x with child pointers Ptr_{left} and Ptr_{right}

   End

   a) Define and solve a recurrence equation for the running time of Slowheap.

   Recall that given a tree H where the heap condition is satisfied except possibly at the root r (that is, $key[r] \geq key[\text{leftchild}(r)]$ and/or $key[r] \geq key[\text{rightchild}(r)]$ and $key[v] \geq key[\text{parent}(v)]$ for all nodes $v \neq r$), we can make H into a heap by performing a Down-Heapify operation on the root r (Down-Heapify on node v swaps element in v with element in one of the children of v and continues down the tree until a leaf is reached or heap order is reestablished).

   Consider the following algorithm FastHeap$(1, n)$ which constructs (a pointer to) H by placing an arbitrary element x from A (the last one) in the root of H, recursively constructing the two sub-heaps below x, and finally performing a Down-Heapify operation on x to make H a heap.

FASTHEAP(i, j)

$Ptr_{\text{left}} =$ FASTHEAP$(i, \lfloor \frac{i+j-1}{2} \rfloor)$

$Ptr_{\text{right}} =$ FASTHEAP$(\lfloor \frac{i+j-1}{2} \rfloor + 1, j-1)$

Let Ptr be pointer to tree consisting of root r containing $x = A[j]$ with child pointers Ptr_{left} and Ptr_{right}

Perform DOWN-HEAPIFY on Ptr

Return Ptr

End

b) Define and solve a recurrence equation for the running time of FASTHEAP.