DESIGN AND ANALYSIS OF ALGORITHMS

GABRIEL MOISE

HT 2019

PROBLEM SHEET 2

Divide and Conquer, cont'd

Question 1

We have the following procedures:

- PARTITION' (A, p, r, x) → takes an array $A[p..r)$ of distinct integers and an element x from A and returns index q with $p \le q < r$ such that we have a new array $A'[p..r)$ with $A'[p..q)$ consisting of elements less than $A[q]$ and $A[q+1..r)$ with elements greater than $A[q]$ ($O(n)$ time)
- MEDIAN' (A, p, r) → takes array $A[p..r)$ of distinct integers and returns the $\lceil (r-p)/2 \rceil$-order statistic of $A[p..r)$ ($O(n)$ time)

* where $n = r-p$

(a) QUICK-SORT' (A, p, r)

Input: $A[p..r)$ array of distinct integers

Output: The sorted array

1. m = MEDIAN' (A, p, r) // The element around which we will partition the array
2. index = PARTITION' (A, p, r, m) // We get two sub-arrays with roughly the same number of elems.
3. QUICK-SORT' (A, p, m) // We sort the left sub-array
4. QUICK-SORT' $(A, m+1, r)$ // And the right sub-array

Let's calculate the complexity:

1. needs $O(n)$ time
2. needs $O(n)$ time
3. needs $T(\lceil \frac{n}{2} \rceil)$ time (as we got two subarrays of size $\lceil \frac{n}{2} \rceil$ at most)
4. needs $T(\lceil \frac{n}{2} \rceil)$ time

Then, $T(n) \le 2T(\lceil \frac{n}{2} \rceil) + O(n)$

By using The Master Theorem, we obtain $T(n) = O(n \log n)$.

(b) SELECT' (A, p, r, i)

Input: $A[p..r)$ array of distinct integers and $i \le r-p$

Output: The i-order statistic of $A[p..r)$

1. m = MEDIAN' (A, p, r)
2. index = PARTITION' (A, p, r, m) // We partition the array in the same manner as before
3. half = $\lceil (r-p)/2 \rceil$ // We compare i to half

4. if (i = half) return m

5. else if (i < half) SELECT'(A, p, half, i) // We need to look in the left subarray

6. else SELECT'(A, half+1, r, i - half - 1) // Or in the right subarray

Let's calculate the time-complexity:

1. needs $O(n)$ time
2. needs $O(n)$ time
3. needs $O(1)$ time
4. needs $O(1)$ time
5. needs $T(\lceil \frac{n}{2} \rceil$ time)
6. needs $T(\lceil \frac{n}{2} \rceil$ time)

$\left.\begin{matrix} 5. \\ 6. \end{matrix}\right\}$! only one of them happens

Then, we have:

$$T(n) \leq T(\lceil \tfrac{n}{2} \rceil) + O(n)$$ (Worst-case scenario → 4. never happens until the end)

By using The Master Theorem, we get $T(n) = O(n)$ => linear-time algorithm

(c) in order to find the i-order statistic using just one call of median', we need to turn the i^{th} smallest element into a median. To do so, we analyse three cases:

1. if $i = \lceil \frac{n}{2} \rceil$, then we do not need to do anything.
2. if $i < \lceil \frac{n}{2} \rceil$, then we need to add $(n-2i)$ elements that are smaller than min (the smallest element in the array, which can be found in n steps) to the array, so that the i^{th} smallest element in the original array becomes the median in the modified array
3. if $i > \lceil \frac{n}{2} \rceil$, we need to add $2i - n$ elements that are greater than max (the biggest element in the array).

The algorithm looks like this:

```
SELECT''(A, p, r, i)

1.  MIN = +∞ ; MAX = -∞ ; n = r-p                       O(1)
2.  for (i <- p until r)                                 } 
3.      if (A[i] < MIN) MIN = A[i]                       } O(n)
4.      if (A[i] > MAX) MAX = A[i]                       } // Computing the smallest and the biggest element of the array
5.  half = ⌈(r-p)/2⌉                                     O(1)
6.  if (i < half)
                                                         O(n)
7.      for (j=0 until n-2i) A[r+j] = MIN-j-1   // Adding n-2i elements less than min
8.      r = r + n - 2i
9.  else if (i > half)
                                                         O(n)
10.     for (j=0 until 2i-n) A[r+j] = MAX + j + 1   // Adding 2i-n elements greater than max
11.     r = r + 2i - n
12. m = MEDIAN'(A, p, r)                                 O(n)
13. return m                                             O(1)
```

Here, the time complexity is clearly linear as all instructions are.

*

Question 2

$n = 800 = 2^5 \cdot 25$

By using the conventional method, we need $2 \cdot 800^3 - 800^2 = 2 \cdot 8^3 \cdot 10^6 - 8^2 \cdot 10^4 = 2^{10} \cdot 10^6 - 2^6 \cdot$
which is greater than 10^9.

With the hybrid method we get: (By using Strassen's trick we have $T(n) = 7T(\lceil \frac{n}{2} \rceil) + 18 \frac{n}{4}$

$$T(800) = 7T(400) + 18 \cdot 400^2$$

$$= 7(7T(200) + 18 \cdot 200^2) + 18 \cdot 400^2 = 7^2 T(200) + 7 \cdot 18 \cdot 200^2 + 18 \cdot 400^2 =$$

$$= \dots =$$

$$= 7^5(25^3 - 2 \cdot 25^2) + (7^4 \cdot 18 \cdot 25^2) + (7^3 \cdot 18 \cdot 50^2) + 7^2 \cdot 18 \cdot 100^2 + 7 \cdot 18 \cdot 200^2 + 18 \cdot 400^2,$$ which is smaller than $6 \cdot 10^8$, so this method is more efficient.

Question 3

We start by supposing that two matrices can be multiplied by performing 32 block multiplications and 144 block additions of $n/4 \times n/4$ matrices.

(a) From this, we have $T(n) \leq 32\,T(\lceil \frac{n}{4} \rceil) + O(n^2)$. By using The Master Theorem we obtain that $T(n) = O(n^{\frac{5}{2}})$. We now want to find $c \in \mathbb{R}$ such that $T(n) \leq c n^{\frac{5}{2}}$, so we will use the recursion tree method: $(n = 4^k, k \geq 0)$

$$T(4^k) = \begin{cases} 32\,T(4^{k-1}) + f(4^k), & k > 0 \\ 1, & k = 0 \end{cases}$$

where $f(4^k) = 144 \cdot (4^{k-1})^2 = 144 \cdot 4^{2k-2} = 9 \cdot 2^{4k}$

We have $T(4^k) = f(4^k) + 32 f(4^{k-1}) + 32^2 f(4^{k-2}) + \dots + 32^{k-1} f(4) + 32^k$

$$= 9 \cdot 2^{4k} + 2^5 \cdot 9 \cdot 2^{4k-4} + 2^{10} \cdot 9 \cdot 2^{4k-8} + \dots + 2^{5k-5} \cdot 9 \cdot 2^4 + 32^k$$

$$= 9 \cdot 2^{4k} + 9 \cdot 2^{4k+1} + 9 \cdot 2^{4k+2} + \dots + 9 \cdot 2^{5k-1} + 2^{5k}$$

$$= 9 \cdot 2^{4k}(1 + 2 + 2^2 + \dots + 2^{k-1}) + 2^{5k}$$

$$= 9 \cdot 2^{4k} \cdot (2^k - 1) + 2^{5k}$$

$$= 9 \cdot 2^{5k} - 9 \cdot 2^{4k} + 2^{5k}$$

$$= 10 \cdot 2^{5k} - 9 \cdot 2^{4k} < c \cdot (4^k)^{\frac{5}{2}}$$ $\Rightarrow$ the constant factor c is ≥ 10.

(b) The conventional algorithm needs $2n^3 - n^2$ operations. We want to find the smallest $n = 4^k$, such that

$$10 \cdot 2^{5k} - 9 \cdot 2^{4k} < 2 \cdot 2^{6k} - 2^{4k} \quad \Big| : 2^{4k} \neq 0$$

$$10 \cdot 2^k - 8 < 2 \cdot 2^{2k}$$

$$2 \cdot 2^{2k} - 10 \cdot 2^k + 8 > 0$$

$$2^{2k} - 5 \cdot 2^k + 4 > 0$$

Let $y = 2^k$ $\Big| \Rightarrow y^2 - 5y + 4 > 0$

$$\Delta = 25 - 16 = 9$$

$$y_{1/2} = \frac{5 \pm 3}{2} = \left\langle \begin{matrix} 4 \\ 1 \end{matrix} \right| \Rightarrow y > 4 \Rightarrow 2^k > 4 \Rightarrow n > 16 \Rightarrow \text{from } n = 64$$

we can use this method. (we use it only for $n = 4^k, k \in \mathbb{N}$)

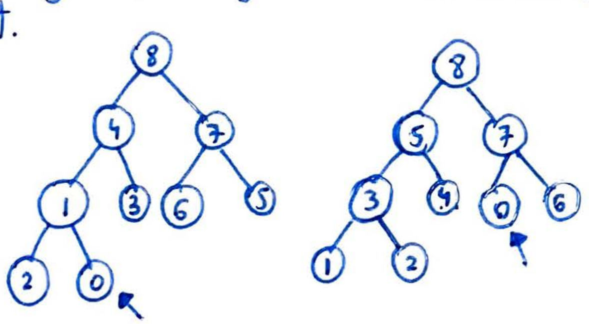
<u>Heaps, heapsort and priority queues</u>

<u>Question 4</u>

(a) Let h be the height of the heap. Then, we have the first $(h-1)$ levels completed, so we have at least $2^0 + 2^1 + 2^2 + \dots + 2^{h-1} = 2^h - 1$ nodes. In order to have height h the heap needs to have at least one leaf on the h^{th} level, so the <u>minimum</u> number of nodes is 2^h and the <u>maximum</u> number is $2^{h+1} - 1$ (when the h^{th} level is complete).

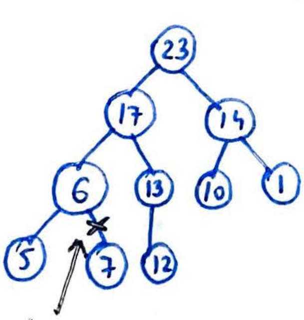
(b) In a max-heap, the smallest element can be a leaf (in any position of the last level), as it can be the child of any node, or it can be on the preceding level, but it must not have any leafs. Any other level would be impossible since the smallest element cannot be a parent.

(Between $\lceil \frac{n+1}{2} \rceil$ and n)



(c) A sorted array is a min-heap as we have for all positions i, with $1 < i \leq A.size$: $A[i] \geq A[\lfloor i/2 \rfloor]$ from the fact that the array is sorted.

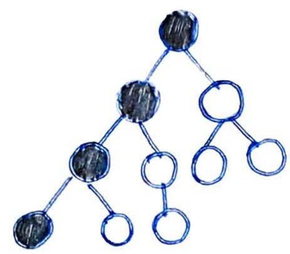
(d) The sequence $[23, 17, 14, 6, 13, 10, 1, 5, 7, 12]$ is represented as a heap as:



here $7 > 6$, so this is not a max-heap

Question 5

In the worst-case scenario, the algorithm always goes in the left branch and because the heap has size n, then there must be a leaf on the position $2^{\lfloor \log_2 n \rfloor}$ to which we'll make $\lfloor \log_2 n \rfloor$ steps: node $2^0 \rightarrow$ node $2^1 \rightarrow$ node $2^2 \rightarrow \dots \rightarrow$ node $2^{\lfloor \log_2 n \rfloor}$. So, in the worst-case scenario, the running-time of MAX-HEAPIFY is $\Omega(\log n)$.



WORST-CASE SCENARIO

Question 6

In order to remove an element from a heap, we swap it with the last element of the heap and then we remove the right-most leaf. Now, in order to maintain the original property of the heap, we need to analyse two cases: (we are analysing the situation for MAX-heap)

1. current node > its parent

   We will then swap the two nodes and continue checking if the node is greater than its new parent until we find that it's not (we go up the heap)

2. current node < one of its children (if the current node = its parent we do nothing)

   We will then swap the two nodes and continue checking if the node is smaller than one of its new children until we find that it's not (we go DOWN the heap)

   (for MIN-heap the procedure is similar, but with reversed signs).

In the worst-case scenario, we need to make $\lfloor \log_2 n \rfloor$ operations if we remove the root and we always go left (like in Question 5).

* Question 7

For both cases the MAKE-MAX-HEAP (or MIN) need $O(n \log n)$ by doing MAX-HEAPIFY, which always needs $O(\log n)$ for every element of the array.

Question 8

We have k sorted lists and we want to merge them into a sorted list. The algorithm for this is :

1. Create a MIN-HEAP with the first elements of the k lists $\Rightarrow O(K)$
2. Return the root of the heap (that is the next element we want to print) and then remove it from the MIN-HEAP by swapping it with the next element from the list it came from $\Rightarrow O(1)$
3. MIN-HEAPIFY the resulting structure from the root $\Rightarrow O(\log k)$
4. Return to 2. and do this until we use all the elements from the k sorted lists.

We run the loop n times and for each loop we need $O(\log k)$ operations, so the time complexity we obtain in the end will be $O(n \log k)$.

(the algorithm is correct since we always have the minimum of the unused elements in the heap and after applying MIN-HEAPIFY it will always be the root (it's either already the root or one of its 2 children)).