Assignment 1 - review questions
SOLUTIONS

1. You are given an array A of n distinct integers.

   (a) You have to determine if there exists a number (not necessarily in A) which can be written as a sum of squares of two distinct numbers from A in two different ways (note: m^2+k^2 and k^2+m^2 counts as a single way) and which runs in time $n^2 \log n$ in the **worst case** performance. Note that the brute force algorithm would examine all quadruples of elements in A and there are $\binom{n}{4} = O(n^4)$ such quadruples. (10 points)

   (b) Solve the same problem but with an algorithm which runs in the **expected time** of $O(n^2)$. (10 points)

   Solution: Go through all of $\binom{n}{2} = \frac{n(n-1)}{2}$ pairs $(A[k], A[m])$, $k < m$, of distinct integers in A; compute the sums $A[k]^2 + A[m]^2$ for all $1 \leq k < m \leq n$ and put them in an array of size $n(n-1)/2$. Sort the array in time $O(n^2 \log_2 n^2 = O(n^2 \log_2 n)$. Go through the sorted array and determine if a number appears in it at least twice (such occurrences would be consecutive).

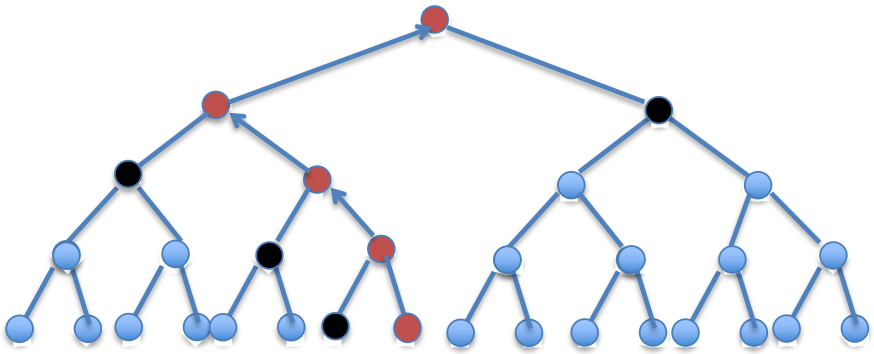
2. Suppose that you bought a bag of n bolts with nuts screwed on them. Your 5 year old nephew unscrewed all the nuts from the bolts and put both the nuts and the bolts back into the bag. The bolts are all of similar quite large size but are actually of many different diameters, differing only by at most a few millimetres, so the only way to see if a nut fits a bolt is to try to screw it on and determine if the nut is too small, if it fits or if it is too large. Design an algorithm for matching each bolt with a nut of a fitting size which runs in the **expected** time $O(n \log n)$. (20 points)

   Solution: Pick a bolt and try all nuts, placing them on three piles: too small, fitting, too large. Now pick a nut from the fitting pile and try all bolts, placing them on three piles: too small, fitting, too large. Screw fitting nuts on each of the fitting bolts. Continue such a process with the piles of too small bolts and too small nuts, as well as with piles of too large bolts and too large nuts. The algorithm runs in time $O(n \log n)$ for exactly the same reason as the QuickSort algorithm.

3. You are given 1024 apples, all of similar but different sizes and a small pan balance which can accommodate only one apple on each side. Your

task is to find the heaviest and the second heaviest apple while making at most 1032 weighings. (20 points)

Solution: We prove a more general statement: you are given a list of 2^n numbers and you can only compare them pairwise, determining which one is larger or if both are equal. We will show that you can find the largest and the second largest number using only 2^n+n-2 comparisons. We now put all of these numbers at the leaves of a complete binary tree, which is a tree in which every node is either a leaf or has exactly two children, see the figure. A complete binary tree with 2^n leaves has 2^n-1

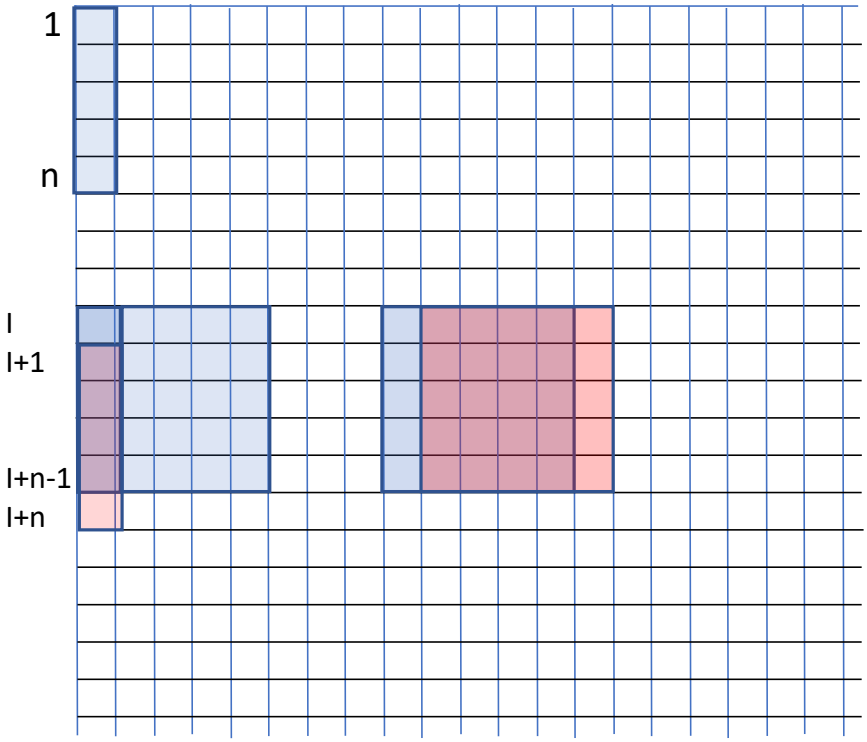


internal nodes and is of depth n. Place all the numbers at the leaves, compare each pair and "promote" the larger element (shown in red) to the next level and proceed in such a way till you reach the root of the tree, which will contain the largest element. Clearly, each internal node is a result of one comparison and there are 2^n-1 many nodes thus also the same number of comparisons so far. Now just note that the second largest element must be among the black nodes which were compared with the largest element along the way - all elements underneath them must be smaller or equal to the elements shown in black. There are n many such elements so finding the largest among them will take $n-1$ comparisons by brute force. In total this is exactly 2^n+n-2 many comparisons.

4. You are in an orchard which has a quadratic shape of size $4n$ by $4n$ with equally spaced trees. You purchased apples from n^2 trees which also form a square, but the owner is allowing to choose such a square anywhere in the orchard. You have a map with the number of apples on each tree. Your task is to chose such a square which contains the

largest total number of apples and which runs in time $O(n^2)$. Note that the brute force algorithm would run in time $\Theta(n^4)$. (20 points)

Solution: The idea here is to note that there is a heavy overlap between such possible squares and to use this fact to compute the number of apples in all of such squares in an efficient way. Referring to the figure below we first consider the first column. Let the number of apples in



cell $C[i, j]$ be $A[i, j]$. We first compute the sum $\alpha(1, 1) = \sum_{k=1}^{n} A[k, 1]$, (corresponding to cells $C[1, 1]$ to $C[n, 1]$ shown in blue in the top left corner of the orchard map). This takes $n - 1 = O(n)$ additions. We then compute the number of apples $\alpha(i, 1)$ in all rectangles $r(i, 1)$ consisting of cells $C[i, 1]$ to $C[i+n-1, 1]$ for all i such that $2 \leq i \leq 3n+1$, starting from $\alpha(1, 1)$ and using recursion

$$\alpha(i + 1, 1) = \alpha(i, 1) - C[i, 1] + C[i + n, 1]$$

This is correct because the rectangle $r(i, 1)$ consisting of cells $C[i, 1]$ to $C[i + n - 1, 1]$ and rectangle $r(i + 1, 1)$ consisting of cells $C[i + 1, 1]$ to $C[i + n, 1]$ in the first column overlap and differ only in the first

square of $r(i)$ and the last square of $r(i+1)$, see the picture. Since each recursion step involves only one addition and one subtraction this can all be done in O(n) many steps in total.

We now perform the same procedure in each column j, thus obtaining the number of apples $\alpha(i,j)$ in every rectangle consisting of cells $C(i,j)$ to $C(i+n-1,j)$, which will take in total $O(n) \times O(n) = O(n^2)$ many steps.

We now for each i such that$1 \leq i \leq 3n+1$ compute $\beta(i,1) = \sum_{p=1}^{n} \alpha(i,p)$; this clearly gives the total number of apples in the square with vertices at cells $C(i,1), C(i,n), C(i+n-1,1)$ and $C(i+n-1,n)$. Each such computation takes $O(n)$ additions so in total doing this for all i it takes $O(n^2)$ many additions.

For each fixed i such that$1 \leq i \leq 3n+1$, starting with $\beta(i,1)$ we can now compute $\beta(i,j)$ for all $2 \leq j \leq 3n+1$ using recursion

$$\beta(i,j+1) = \beta(i,j) - \alpha(i,j) + \alpha(i,j+n-1)$$

because the two adjacent squares overlap, except for the first column of the first square and the last column of the second square. Each step of recursion takes only one addition and one subtraction so the whole recursion takes $O(n)$ many steps. Thus, all recursions for all $1 \leq i \leq 3n+1$ takes $O(n^2)$ many operations. We finally find the largest $\beta(i,j)$.

5. Determine if $f(n) = O(g(n))$ or $g(n) = O(f(n)$ or both (i.e., $f(n) = \Theta(g(n))$) or neither of the two, for the following pairs of functions

   (a) $f(n) = (\log_2(n))^2$; $\quad g(n) = \log_2(n^{\log_2 n})^2$; $\quad$ (6 points)

   (b) $f(n) = n^{10}$; $\quad g(n) = 2^{\sqrt[10]{n}}$; $\quad$ (6 points)

   (c) $f(n) = n^{1+(-1)^n}$; $\quad g(n) = n$. $\quad$ (8 points)

   Solution:

   (a) Note that $\log_2(n^{\log_2 n})^2 = \log_2(n^{2\log_2 n}) = 2\log_2 n \log_2 n = 2(\log_2 n)^2 = \Theta((\log_2(n))^2)$

   (b) We show that eventually $n^{10} < 2^{\sqrt[10]{n}}$. Taking the logs of both sides, since $\log_2 n$ is a monotonically increasing function, it is enough to prove that $\log n^{10} = 10 \log n < \log_2 2^{\sqrt[10]{n}} = \sqrt[10]{n}$. To show that eventually $10 \log n < \sqrt[10]{n}$ we show that $\lim_{n\to\infty} \log n / \sqrt[10]{n} = 0$.

Since both the numerator and the denominator go to infinity, we can apply the L'Hôpital rule and get

$$\lim_{n\to\infty} \frac{\log n}{\sqrt[10]{n}} = \lim_{n\to\infty} \frac{(\log n)'}{(\sqrt[10]{n})'} = \frac{\frac{1}{n}}{\frac{1}{10}n^{-9/10}} = 10\frac{1}{n^{1/10}} \to 0$$

(c) Just note that for all even n, $f(n) = n^2$ which grows much faster than $g(n) = n$. However for all odd n we have $f(n) = n^0 = 1$ so for odd n $g(n) = n$ grows much faster than $f(n)$. Thus, neither $f(n) = O(g(n))$ nor $g(n) = O(f(n))$.