EECS 586 Algorithms **Winter 2022**

Homework 1

Instructor: Prof. Thatchaphol Saranurak
GSI: Shang-En Huang
GSI: Dingyu Wang

Due on January 21, 2022

If you are asked to devise, describe, or show an algorithm that solves the problem, you MUST also provide the correctness proof and the runtime analysis (in terms of asymptotic functions unless specified otherwise.) We grade your homeworks according to the clarity, not just correctness.

1 Recurrence Relations (4 points)

Solve the following Recurrence Relations. It is allowed to cite and use the master theorem whenever applicable. Write down the answer as tight (and as simple) as you can in asymptotic notations. You may assume that $T(x) = 1$ for all $x \leq 2$ (some constant) and the recurrence relations are defined whenever $x > 2$. You may use the fact that $T(n)$ is non-decreasing and all logarithms here are base-2.

(a) (0.5 points) $T(n) = 2T(\frac{n}{2}) + n \log n$

(b) (0.5 points) $T(n) = 2T(\frac{n}{2}) + n \cdot \frac{\log \log n}{\log n}$

(c) (0.5 points) $T(n) = T(\frac{n}{2}) + T(\frac{n}{3}) + n^2$

(d) (0.5 points) $T(n) = 2T(\sqrt{n}) + 1$

(e) (0.5 points) $T(n) = \sqrt{n}T(\sqrt{n}) + n$

(f) (0.5 points) $T(n) = T(n-1) + T(n-4) + 1$

(g) (1 point) $T(n) = T(n-1) + 2T(n/2) + 1$

2 Dominating Pairs (4 points)

You are given n vectors $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ in d-dimensional space $\mathbb{R}^d$. For simplicity let's assume that all entries in all vectors are distinct. We say that the vector $\mathbf{a} = (a_1, a_2, \ldots, a_d)$ dominates another vector $\mathbf{b} = (b_1, b_2, \ldots, b_d)$ if and only if $a_i > b_i$ for all i. This domination relation is denoted by $a \succ b$.

Let the set S be the collection of all dominating pairs $\{(i, j) : \mathbf{x}_i \succ \mathbf{x}_j\}$. The goal is to compute (and output) this set S.

(a) (2 point) Suppose $d = 2$, devise an algorithm that computes the set S in $O(|S| + n \log n)$ time.

(b) (2 points) For any d, devise an algorithm that computes the set S in $O(|S| + n \log^{d-1} n)$ time. Compare this runtime with naive algorithm that runs in $O(|S| + n^2 d)$ time: assume $|S|$ is small, in what regime of parameters d and n the devised algorithm become more efficient?

Remarks. One can actually achieve $O(|S| + n \log^{d-2} n)$ runtime whenever $d \geq 3$. This requires an advanced technique called *fractional cascading* with a dynamic range-reporting data structure.

💬 On the scale 1 to 5, how fun do you think for this problem? How hard do you think for this problem?

3 In Word Parallelism (4 points)

Modern computers nowadays have *machine words* of 64-bits or sometimes 128-bits. Suppose you have a really cool machine with word size W-bits. For simplicity we assume $W = 2^w$ for some even integer w. On this cool machine, you can perform any bitwise operations (shift left, shift right, bitwise AND, bitwise OR, bitwise XOR, bitwise negation) in constant time. You can also perform additions and subtractions (not multiplications) on machine words (treating them as unsigned integers and ignore all overflow bits) in constant time.

(a) (1 point) Given a machine word $x \neq 0$, devise an algorithm that extracts the least significant bit (LSB) in $O(1)$ time. That is, all the bits that are higher than LSB should be cleared to 0. For example, $LSB(\texttt{0010101001101000}) = \texttt{0000000000001000}$.

Hint: what happened to LSB if you subtract x by 1?

(b) (1 point) Given a machine word $x \neq 0$, devise an algorithm that extracts the most significant bit (MSB) in $O(\log W)$ time. For example, $MSB(\texttt{0010101001101000}) = \texttt{0010000000000000}$.
Remarks. If constant time multiplications are allowed, you can actually achieve $O(1)$ time.

Hint: binary search!

(c) (1 point) A machine word x can represent a set $X \subseteq \{0, 1, \ldots, W-1\}$. Given a machine word x that represents the set X, devise an algorithm that computes $|X|$ in $O(\log W)$ time. This quantity is often called *population count* or *Hamming weight*. For example, $POPCOUNT(\texttt{0010101001101000}) = \texttt{0000000000000110}$ (this is 6 in binary).

Hint: divide and conquer!

(d) (1 point) A machine word x can represent a boolean matrix A of size $2^{w/2} \times 2^{w/2}$ (in the row major representation), where $A[i, j]$ can be accessed at the $(i(w/2) + j)$-th bit. Given a machine word x that represents A, devise an algorithm that transposes A in $O(\log W)$ time. For example, when $w = 4$ we have $TRANSPOSE(\texttt{0010101001101000}) = \texttt{0101001011100000}$.

💬 On the scale 1 to 5, how fun do you think for this problem? How hard do you think for this problem?

4 Building Highways (4 points)

There are n junctions on a super-duper long road (thinking of US Route 66). The junctions are numbered $1, 2, \ldots,$ and n from the west to the east. At each junction there is a stop sign so that if any vehicle passes through that junction it must stop and then start again. The government wants

to build some highways that connect pairs of junctions. Any vehicle may get on a highway from one of its endpoint junction but this vehicle cannot leave highway until it arrives another endpoint. Of course, there is no stop sign on any highway.

The goal of building these highways, is to minimize the number of stops that are required when traveling from any junction to any junctions to the east. The vehicle must stop at the endpoint junction before getting on another highway (or using the original road).

(a) (1 point) Suppose that all roads and highways are bidirectional. Show that you can add $O(n)$ highways so that traveling between any two junctions requires only 1 stop.

(b) (1 point) Suppose that all roads and highways are uni-directional (one-way roads toward east). Moreover, any vehicle can only move east all the time.

Show that you can add $O(n)$ highways so that traveling between any two junctions requires only $O(\log n)$ stops.

(c) (2 point) Suppose that all roads and highways are uni-directional (one-way roads toward east). Moreover, any vehicle can only move east all the time.

Show that you can add $O(n \log^* n)$ highways so that traveling between any two junctions requires only $O(1)$ stops. Here $\log^* n$ denotes the *iterated logarithm*, which is the minimum positive integer k such that

$$\underbrace{\log \log \cdots \log}_{k}(n) \leq 1.$$

(You get 1.5 points if your construction achieves $O(1)$ stops by adding $\Theta(n \log \log n)$ highways. You get 0 points if your construction achieves $O(1)$ stops by adding $\Omega(n \log n)$ highways.)

💬 On the scale 1 to 5, how fun do you think for this problem? How hard do you think for this problem?

5 Deterministic Selection (4 points)

Notice that after working out this problem you have (slightly) improved the state-of-the-art results of Ke Chen and Adrian Dumitrescu (See https://dc.uwm.edu/etd/2652/ and the updated arXiv version to their original WADS 2015 paper), isn't it cool!

In the class we have discussed the famous $O(n)$ time deterministic selection BFPRT algorithm. Now we are trying to extend this idea into groups of various sizes.

(a) (1 point) Show that if we change the group size of the BFPRT algorithm into groups of 7, then the algorithm still runs in $O(n)$ time.

Now, suppose we define the median of a group of 4 numbers to become the 2nd smallest number among the group. That is, the median of $\{3, 9, 102, 514\}$ is 9. In general, we define the median of a group of $2t$ (even) numbers to become the t-th smallest number among the group. From now on, assume that we have applied the above definitions and use groups of size 4 to the BFPRT algorithm.

(b) (1 point) Show that if we are selecting the median $k = \lceil n/2 \rceil$, after 3 levels of recursion the problem size becomes at most $45n/128$.

(c) (1 point) Let us define two different function $S(n)$ and $T(n)$, where $T(n)$ is the upper bound of runtime that solves the selection problem for an *arbitrary* rank k. We use $S(n)$ to denote the upper bound of runtime that solves the selection for $k = \lceil n/2 \rceil$ being exactly the median. Establish an upper bound of $S(n)$ using T functions and n.

(d) (1 point) This is the brandly new part! Nobody in the world have found it so far except you! (...and the teaching staff, of course.) Write down the recurrence relation of $T(n)$ using both $S(n)$ and $T(n)$, then prove that both $S(n) = O(n)$ and $T(n) = O(n)$.

Remarks. In conclusion, we do not need to use the shifting target algorithm from Chen and Dumitrescu, we get linear time automatically! (Caution: it is very important to analyze the leading constants hiding in big-O and you have to be very careful.)

💬 On the scale 1 to 5, how fun do you think for this problem? How hard do you think for this problem?

6 Hamming Distance (4 points)

In this problem we use the set Σ to denote the alphabet (collection of all available *characters*) and we use Σ^* to denote the set of all strings. Suppose you are given a long *text* $T[0..n-1] \in \Sigma^n$ and a relatively short *pattern* $P[0..m-1] \in \Sigma^m$ (with $m < n$.) Define *Hamming distance* over two strings X and Y of the same length to be the number of locations where two strings are different:

$$ham(X, Y) \stackrel{\text{def}}{=} |\{i : x[i] \neq y[i]\}|.$$

Our goal is to compute the Hamming distances between the text and the pattern in all shifts. That is, to compute an array $A[0, 1, 2, \ldots, n-m]$ where $A[i] = ham(T[i..i+m-1], P)$ for all i.

(a) (2 points) Devise an algorithm that computes array A in $O(|\Sigma| n \log n)$ time.

(b) (1 point) Devise an algorithm that computes array A in $O(|\Sigma| n \log m)$ time. You can reuse part (a) as a subprocedure. (Hint: chop the text into smaller pieces and run part (a) on each piece.)

(c) (1 point) Assume that the characters are just integers: $\Sigma = \{1, 2, \ldots, U\}$, with $U \leq n$. Devise an algorithm that computes array A in $O(n\sqrt{m \log m})$ time. (Hint: there are at most $\sqrt{m/\log m}$ different characters with frequency more than $\sqrt{m \log m}$ in the pattern. Apply part (b) only to the set of high-frequency characters.)

💬 On the scale 1 to 5, how fun do you think for this problem? How hard do you think for this problem?