Introduction to Algorithms **CS 4820, Spring 2019**	**Solution Set 5**

(1) *(15 points) Let $X_1, X_2, \ldots, X_k$ be a independent random variables taking values in the range $\{0, 1, \ldots, n-1\}$. Their sum $X = X_1 + \cdots + X_k$ takes values in the range $\{0, \ldots, k(n-1)\}$. Suppose we are given an input that specifies:*

1. **the distribution of each** X_i, *expressed in the form of a two-dimensional array P such that $P[i, j]$ denotes the probability that $X_i = j$.*
2. **an interval** $[a, b]$, *such that $0 \le a \le b \le k(n-1)$.*

Given this data, we are interested in computing the probability that $a \le X \le b$.

(1a) (5 points) *Design a dynamic programming algorithm that computes, for every pair i, j, the quantity $q_{ij} = \Pr(X_1 + \cdots + X_i = j)$, and then outputs the sum $\sum_{j=a}^{b} q_{kj}$.*

You may omit the proof of correctness, but you should analyze the running time of this dynamic programming algorithm.

(1b) (10 points) *Design an algorithm that computes $\Pr(a \le X \le b)$ in time $O(k\, n \log(k) \log(kn))$.*

For this part of the problem, include both the running time analysis and the proof of correctness.

HINT: If Y and Z are independent random variables taking values in $\{0, 1, \ldots, n-1\}$, show that the probability distribution of their sum $Y+Z$ can be computed as the convolution of two vectors representing the probability distributions of Y and of Z.

Solution. First we prove the fact stated in the hint.

Lemma 1. *Suppose Y and Z are independent random variables taking values in $\{0, 1, \ldots, n-1\}$. Let $y = (y_0, y_1, \ldots, y_{n-1})$ and $z = (z_0, z_1, \ldots, z_{n-1})$ denote vectors representing the probability distributions of Y and Z; in other words for $0 \le i < n$,*

$$y_i = \Pr(Y = i), \quad z_i = \Pr(Z = i).$$

Then the probability distribution of $Y + Z$ is represented by the convolution $y * z$; in other words for $0 \le j < 2n - 1$,

$$\Pr(Y + Z = j) = \sum_{i=0}^{i} y_i z_{j-i}.$$

Proof. The event that $Y + Z = j$ is the union of the disjoint events $\mathcal{E}_{ij} = \{Y = i, Z = j - i\}$ where i ranges from 0 to i. Hence

$$\Pr(Y + Z = j) = \sum_{i=0}^{j} \Pr(\mathcal{E}_{ij}) = \sum_{i=0}^{j} \Pr(Y = i) \cdot \Pr(Z = j - i) = \sum_{i=0}^{j} y_i z_{j-i}$$

where the second equation is due to the independence of Y and Z, and the third equation follows from the definition of the vectors y and z. $\square$

(1a) The dynamic programming algorithm is as follows. The correctness of the dynamic programming algorithm is proven by induction on i. The induction hypothesis is that for $0 \le j \le k(n-1)$, the dynamic

---

Initialize $Q[0,0] = 1$ and $Q[0,j] = 0$ for $j = 1, 2, \ldots, k(n-1)$.
for $i = 1, \ldots, k$ **do**
  for $j = 0, \ldots, k(n-1)$ **do**
    $Q[i,j] = \sum_{\ell=0}^{j} Q[i-1,\ell] \cdot P[i, j-\ell]$
  end for
end for
Return $Q[k,a] + Q[k,a+1] + \cdots + Q[k,b]$.

---

programming table entry $Q[i,j]$ is equal to the probability q_{ij} defined in the problem statement. The induction step is an application of Lemma 1.

The running time is $O(k^3n^2)$ because the dynamic programming table has $O(k^2n)$ entries, and computing one entry involves calculating a sum of $O(kn)$ terms, where each term of the sum can be calculated in $O(1)$ time.

(1b) We first present an algorithm with running time $O(k^2n\log(kn))$ and then explain how to improve the running time to $O(k\,n\log(k)\log(kn))$.

The idea of the $O(k^2n\log(kn))$ algorithm is to replace the inner loop over j in the dynamic programming algorithm with the fast convolution algorithm from Section 5.6 of the textbook. Each loop iteration

---

Let $d = 1 + k(n-1)$.
Initialize d-dimensional vector $Q_0 = (1, 0, \ldots, 0)$.
for $i = 1, \ldots, k$ **do**
  Let P_i denote the d-dimensional vector $(P[i,0], P[i,1], \ldots, P[i,n-1], 0, 0, \ldots, 0)$.
  Let $Q_i = Q_{i-1} * P_i$. Compute Q_i using the fast convolution algorithm based on the FFT.
end for
Return $Q_k[a] + Q_k[a+1] + \cdots + Q_k[b]$.

---

requires computing the convolution of two d-dimensional vectors, hence it runs in $O(d\log d)$ time. The total running time is therefore $O(kd\log d)$; substituting $d = O(kn)$ this implies a running time bound of $O(k^2n\log(kn))$.

To improve the running time to $O(kn\log(k)\log(kn))$, we replace the loop over $i = 1, \ldots, k$ with a divide-and-conquer strategy that partitions the variables $X_1, \ldots, X_k$ into two equal-sized groups, computes the distribution of the sum in each group, and merges the two distributions using the fast convolution algorithm. The benefit of this divide-and-conquer strategy is that near the bottom of the "recursion tree", e.g. when computing the distribution of the sum $X_1 + X_2$, we are convolving vectors whose dimensionality is much less than kn, which results in some running time savings.

Let 2^ℓ denote the smallest power of 2 greater than or equal to k.
for $i = 1, \ldots, 2^\ell$ **do**
  if $i \le k$ **then**
    Let $R_{i,0}$ denote the n-dimensional vector $(P[i,0], P[i,1], \ldots, P[i,n-1])$
  else
    Let $R_{i,0} = (1, 0, \ldots, 0)$.
  end if
end for
for $j = 1, 2, \ldots, \ell$ **do**
  for $i = 1, 2, \ldots, 2^{\ell-j}$ **do**
    Compute $R_{i,j} = R_{2i-1,j-1} * R_{2i,j-1}$ using the fast convolution algorithm based on the FFT.
  end for
end for
Return $R_{1,\ell}[a] + R_{1,\ell}[a+1] + \cdots + R_{1,\ell}[b]$.

The correctness of the algorithm follows immediately from the following lemma.

Lemma 2. *For $0 \le j \le \ell$ and $1 \le i \le 2^{\ell-j}$, let W_{ij} denote the random variable*

$$W_{ij} = \sum_{2^j(i-1) < m \le \min\{k, 2^j i\}} X_m.$$

The vector $R_{i,j}$ computed by the algorithm is $(2^j n)$-dimensional and it encodes the probability distribution of W_{ij}.

Proof. The proof is by induction on j. In the base case $j = 0$ the sum defining the variable W_{ij} either consists of a single term X_i, if $1 \le i \le k$, or it is an empty sum if $i > k$. Hence $W_{i0} = X_i$ if $i \le k$ and $W_{i0} = 0$ if $i > k$, and in both cases the distribution of W_{i0} is correctly encoded by the n-dimensional vector $R_{i,0}$. This finishes the base case.

For the induction step, the induction hypothesis asserts that each of the vectors $R_{2i-1,j-1}$ and $R_{2i,j-1}$ is $(2^{j-1}n) - \mathit{dimensional}$ so their convolution is a vector of dimension $2 \cdot 2^{j-1}n = 2^j n$, as claimed. Applying Lemma 1 along with the induction hypothesis, we find that the vector $R_{i,j}$ encodes the probability distribution of the random variable

$$\begin{aligned} W_{2i-1,j-1} + W_{2i,j-1} &= \left(\sum_{2^{j-1}(2i-2) < m \le 2^{j-1}(2i-1)} X_m \right) + \left(\sum_{2^{j-1}(2i-1) < m \le 2^{j-1}(2i)} X_m \right) \\ &= \sum_{2^{j-1}(2i-2) < m \le 2^{j-1}(2i)} X_m \\ &= \sum_{2^j(i-1) < m \le 2^j i} X_m \;=\; W_{i,j} \end{aligned}$$

as claimed. □

Finally, to analyze the running time of the algorithm, note that in outer loop iteration j, we perform $2^{\ell-j}$ convolution operations on vectors of dimension $2^j n$. The running time of one such convolution is

$$O(2^j n \log(2^j n)) \le O(2^j n \log(kn))$$

since $2^j < 2k$. The running time of $2^{\ell-j}$ such convolutions is therefore $O(2^\ell n \log(kn)) = O(kn \log(kn))$, where we have used the fact that $k \le 2^\ell < 2k$. Finally, observing that the number of iterations of the outer loop is $\lceil \log_2(k) \rceil$, we obtain a running time bound of $O(kn \log(k) \log(kn))$ for all of the outer loop iterations combined. The initialization step and the final step of summing $R_{1,\ell}[a]+R_{1,\ell}[a+1]+\cdots+R_{1,\ell}[b]$ both take $O(kn)$ time, which does not affect the asymptotic running time bound.

(2) *(15 points)* Given as input a list of n points $L = \{(a_1, b_1), \ldots, (a_n, b_n)\}$ on the real plane, your task is to compute the largest rectangle (in terms of area) that can be formed by selecting two points from L, one representing the bottom-left vertex of the rectangle and the other representing the top-right vertex. For simplicity, assume that all the a_i's and b_i's are distinct real numbers.

(2a) *(3 points)* Define two lists of points BL and TR in the following way:

$$BL = \{(a_i, b_i) \in L : \text{for any } j \neq i, \text{ either } a_i < a_j \text{ or } b_i < b_j\}$$

and

$$TR = \{(a_i, b_i) \in L : \text{for any } j \neq i, \text{ either } a_i > a_j \text{ or } b_i > b_j\}.$$

Prove that there exists a rectangle with largest area (using points from L) that has its bottom-left vertex in BL and top-right vertex in TR. Provide an $O(n \log n)$ time algorithm to compute BL and TR. You must output each of the two lists BL and TR by sorting them according to the x-coordinates of the points (in increasing order). **You don't have to provide proof of correctness of your algorithms. You do have to analyze run-time of the algorithms you provide.**

(2b) *(2 points)* Let (a_i, b_i) and (a_j, b_j) be points in BL such that $a_i < a_j$. Further, let (a_k, b_k) and (a_ℓ, b_ℓ) be points in TR such that $a_k < a_\ell$. Define $\Delta_{e,f}$ to be the area of the rectangle using (a_e, b_e) as the bottom-left vertex and (a_f, b_f) as the top-right vertex, where $e \in \{i, j\}$ and $f \in \{k, l\}$. Prove that $\Delta_{i,k} + \Delta_{j,\ell} > \Delta_{i,\ell} + \Delta_{j,k}$.

(2c) *(10 points)* Design an algorithm that runs in time $O(n \log n)$ to compute the largest rectangle (in terms of area) that can be formed by selecting the bottom-left vertex from BL and the top-right vertex from TR. The output of the algorithm should be the area of the largest rectangle.

Solution. We will use the following notation for convenience: we will say the rectangle formed by (i, j) to mean the rectangle formed by the points (a_i, b_i) and (a_j, b_j) as the bottom-left and top-right vertices, respectively (whenever the rectangle is well defined).

(2a) Let (i, j) form an optimal rectangle. Suppose if possible $(a_i, b_i) \notin BL$. Thus there exists (a_k, b_k) such that $a_i > a_k$ and $b_i > b_k$. The area of recatangle formed by (k, j) is $(a_j - a_k)(b_j - b_k)$. Using the fact that $a_i > a_k$ and $b_i > b_k$, it follows that $(a_j - a_k)(b_j - b_k) > (a_j - a_i)(b_j - b_i)$. Noticing that the quantity on the right in the inequality is the area of the rectangle (i, j), it contradicts the optimality of (i, j). Thus, (a_i, b_i) must be in BL. A similar argument shows that $(a_j, b_j) \in TR$. We use the following algorithm to compute BL.

Algorithm 1 computeBL

Sort L in increasing order of the x-coordinate. Let L_x denote this sorted list.
$y_{min} \leftarrow 0$, $BL \leftarrow \emptyset$.
for i from 1 to n **do**
  Let (a, b) be the i'th point in L_x.
  if $b \leq y_{min}$ **then**
    $BL \leftarrow BL \cup \{(a, b)\}$
  end if
  $y_{min} = \min(y_{min}, b)$
end for
Output BL.

Algorithm 2 computeTR

Sort L in decreasing order of the x-coordinate. Let L_x denote this sorted list.
$y_{max} \leftarrow 0$, $TR \leftarrow \emptyset$.
for i from 1 to n **do**
  Let (a, b) be the i'th point in L_x.
  if $b \geq y_{max}$ **then**
    $BL \leftarrow BL \cup \{(a, b)\}$
  end if
  $y_{max} = \max(y_{max}, b)$
end for
Output TR after sorting it in increasing order of x-coordinates.

The sorting step to produce L_x requires $O(n \log n)$ time. Further, each iteration of the loop takes $O(1)$ time. Thus the above algorithm runs in time $O(n \log n)$.

An identical argument also shows that the following algorithm runs in time $O(n \log n)$.

Proof of correctness is not required for HW submission. We supply it here for completeness. We prove the correctness of the algorithm computeBL by using induction on the loop counter i. Let $L_{x,i}$ denote the first i points in the list L_i. Further let BL_i be the set BL at the end of iteration i and $y_{max,i}$ be the value of y_{max} at the end of the i'th iteration. The following is our induction hypothesis: For $i \in \{1, \ldots, n\}$,

$$BL_i = \{(a, b) \in L_{x,i} : \text{ for any } (c, d) \in (L_{x,i} \setminus \{(a, b)\}), \text{ either } a < c \text{ or } b < d\} \text{ and}$$

$$y_{min,i} = \min\{b : \text{for some } a \in \mathbb{R}, (a, b) \in L_{x,i}\}$$

The base case for $i = 1$ is direct from the fact that L_x is sorted in increasing order of x-coordinates (the assertion about $y_{max,1}$ is trivial since there is just 1 point). Now suppose the the induction hypothesis is true for $i - 1$, and we will prove it for i, where $i > 1$. Suppose the i'th point in L_x is (a, b). We start by observing that $BL_{i-1} \subseteq BL_i$. This follows since $L_{x,i} = L_{x,i-1} \cup \{(a, b)\}$ and that a is larger than the x-coordinate of any point in $L_{x,i-1}$.

Now consider the case when (a, b) is included in BL_i. Then it must be that $b < y_{min,i-1}$, i.e., the y-ccordinate of (a, b) is smaller than any the y-coordinate of any point appearing before (a, b) in L_x. Thus, it follows that $(a, b) \in BL_i$, and hence BL_i satisfies the induction hypothesis. The case when (a, b) is not included in BL_i implies that there is some $(c, d) \in L_{x,i-1}$ such that $d < b$. But since $c < a$, it follows from definition that $(a, b) \notin BL_i$. This completes the inductive proof in this case as well.

Finally, observe the induction hypothesis with $i = n$ implies that the output indeed computes BL.

We state the algorithm to compute TR but don't include the proof of correctness since it is almost identical to the above argument.

(2b) Note that since (a_i, b_i) and (a_j, b_j) are both points in BL and $a_i < a_j$, it must be that $b_i > b_j$. Similarly, since (a_k, b_k) and (a_ℓ, b_ℓ) are both points in TR and $a_k < a_\ell$ it must be that $b_k > b_\ell$. We have $\Delta_{e,f} = (a_f - a_e)(b_f - b_e)$, where $e \in \{i, j\}$ and $f \in \{k, \ell\}$. Plugging this in, we have

$$\begin{aligned}\Delta_{i,k} + \Delta_{j,\ell} - \Delta_{i,\ell} - \Delta_{j,k} &= (a_k - a_i)(b_k - b_i) + (a_\ell - a_j)(b_\ell - b_j)\\ &\quad - (a_\ell - a_i)(b_\ell - b_i) - (a_j - a_k)(b_j - b_k)\\ &= (a_\ell - a_k)(b_i - b_j) + (a_j - a_i)(b_k - b_\ell) \qquad \text{(using algebraic manupulations)}\\ &> 0,\end{aligned}$$

where the last inequality follows using the facts that $a_\ell > a_k, b_i > b_j, a_j > a_i$ and $b_k > b_\ell$.

(2c) We will use a divide-and-conquer strategy to design the algorithm. The crucial observation is the following: Let BL and TR denote the lists computed above (recall that they are sorted according to the lists are sorted in increasing order of their x-coordinates), and let q, r denote their respective lengths. We introdiuce some notation: let (c_i, d_i) denote the i'th point in BL and (e_i, f_i) denote the i'th the point in the list TR. Further, for $1 \le i \le q$, define $m(i)$ to be the value in $\{1, \ldots, r\}$ which maximizes the area of the rectangle formed by (c_i, d_i) as the bottom-left vertex and $(e_{m(i)}, f_{m(i)})$ as the top-right vertex. The following simple claim lets us develop a divide-and-conquer strategy.

Claim 1 For $1 \le i < j \le q$, $m(i) \le m(j)$.

Proof Suppose if possible that $m(i) > m(j)$. It follows that by **(2b)** that $\Delta_{i,m(i)} + \Delta_{j,m(j)} < \Delta_{i,m(j)} + \Delta_{j,m(i)}$. Thus at least one of the inequalities hold: (i) $\Delta_{i,m(i)} < \Delta_{i,m(j)}$, (ii) $\Delta_{j,m(j)} < \Delta_{j,m(i)}$, which contradicts the definition of $m()$. $\square$

The divide-and-conquer strategy is the following: Let $MaxArea((i, j), (k, \ell))$ be a function that takes two lists that computes the optimal rectangle with the bottom-left vertex in the range $[i, j]$ in BL and top-right vertex in the range $[k, \ell]$ in TR. To compute $MaxArea(BL, TR)$, first compute $m(w)$ for $w = \lfloor \frac{i+j}{2} \rfloor$. Output the max of $MaxArea((i, w), (k, m(w))$ and $MaxArea((w + 1, j), (m(w), \ell))$.

We now provide pseudocode for the above strategy. We assume that BL is a list of size q and TR is a list of size r.

Algorithm 3 $MaxArea((i, j), (k, \ell))$

```
if i = j or i + 1 = j then
   Output the max-area rectangle by brute-force (i.e, going over all possible pairs).
else
   w ← ⌊(i+j)/2⌋.
   Compute m(w).
   Output the maximum of MaxArea((i, w), (k, m(w)) and MaxArea((w + 1, j), (m(w), ℓ)).
end if
```

Algorithm 4 $Largest - Rect(BL, TR)$

```
Output MaxArea((1, q), (1, r))
```

We now analyze the run time of $MaxArea((i, j), (k, \ell))$, which we denote by $T(\alpha, \beta)$, where $\alpha = j - i + 1, \beta = \ell - k + 1$. We observe that $m(w)$ is some number in the range $k, \ldots, \ell$ and can be computed in time $O(\beta)$. Thus, $T(\alpha, \beta) = T(\lfloor \frac{\alpha}{2} \rfloor, \gamma) + T(\lceil \frac{\alpha}{2} \rceil, \beta - \gamma) + O(\alpha + \beta)$, for some $\gamma \in \{1, \ldots, \beta\}$, which can be solved to get $T(\alpha, \beta) = O((\alpha + \beta) \log \alpha)$. Thus the running time of $Largest-Rect(BL, TR)$ is $O((q + r) \log q)$, which gives us an $O(n \log n)$ time algorithm (since $q, r \le n$).

The proof of correctness of $MaxArea((i, j), (k, \ell))$ follows by an inductive argument on the quantity $\alpha + \beta$, where $\alpha = j - i + 1, \beta = \ell - k + 1$. The base case of $\alpha + \beta = 2$ is direct, since it must be that $i = j$

and $k = \ell$. We now assume the correctness of the algorithm for all valid inputs $(i', j'), (k', \ell')$ such that $(j' - i') + (\ell' - k') < \alpha + \beta$, and prove correctness for all inputs $(i, j), (k, \ell)$ such that $(j - i) + (\ell - k) = \alpha + \beta$. By inductive hypothesis, it follows that $MaxArea((i, w), (k, m(w))$ output the largest rectangle formed with the bottom-left vertex in the range $[i, w]$ in BL and the top-right vertex in the range $[k, m(w)]$ in TR. Similarly, $MaxArea((w+1, j), (m(w)+1, \ell))$ output the largest rectangle formed with the bottom-left vertex in the range $[w+1, j]$ in BL and the top-right vertex in the range $[m(w)+1, \ell]$ in TR. It now follows as a direct application of Claim 1 that the largest rectangle formed with the bottom-left vertex in the range $[i, j]$ in BL and the top-right vertex in the range $[k, \ell]$ in TR must be formed by either (i) the bottom-left vertex in the range $[i, w]$ in BL and the top-right vertex in the range $[k, m(w)]$ in TR, or (ii) the bottom-left vertex in the range $[w + 1, j]$ in BL and the top-right vertex in the range $[m(w) + 1, \ell]$. The correctness is of our algorithm is now direct.