COMP3821
Assignment 2

Charlie Bradford z5114682

April 16, 2018

1. (a) In two n-degree polynomials, P_a and P_b, the value of the product at a certain point, $P_c(x)$, can be found by multiplying $P_a(x)$ and $P_x(x)$. Further, P_c has at most $2n-1$ coefficients, so finding the value of P_c at $2n-1$ different points is enough to determine the exact values of all the different coefficients. When n is sufficiently large, any number that does not have a modulus of exactly 1 can become impossible to represent, so to prevent this the points tested are $2n-1^th$ roots of unity. We then pad P_a and P_b to length $2n-1$ with zeroes. We use the FFT to find the DFT ($P(x)$ for all $x = \omega^i_{2n-1}, 0 \leq i \leq 2n-2$) of both polynomials. We multiply these sequences point wise to find the value of P_c at all these points. Then we use a series of linear equations (or more easily a matrix multiplication, see below) to figure out each of the coefficients of P_c.

$$\begin{bmatrix} C_0 \\ C_1 \\ C_2 \\ \vdots \\ C_{2n} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_{2n-1} & \omega^2_{2n-1} & \dots & \omega^{2n}_{2n-1} \\ 1 & \omega^2_{2n-1} & \omega^{2*2}_{2n-1} & \dots & \omega^{2*2n}_{2n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{2n}_{2n-1} & \omega^{2n*2}_{2n-1} & \dots & \omega^{2n*2n}_{2n-1} \end{bmatrix}^{-1} \begin{bmatrix} P_c(1) \\ P_c(\omega^1_{2n}) \\ P_c(\omega^2_{2n}) \\ \vdots \\ P_c(\omega^{2n-2}_{2n-1}) \end{bmatrix}$$

$$= \frac{1}{n} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega^{-1}_{2n-1} & \omega^{-2}_{2n-1} & \dots & \omega^{-2n}_{2n-1} \\ 1 & \omega^{-2}_{2n-1} & \omega^{-(2*2)}_{2n-1} & \dots & \omega^{-(2*2n)}_{2n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(2n)}_{2n-1} & \omega^{-(2n*2)}_{2n-1} & \dots & \omega^{-(2n*2n)}_{2n-1} \end{bmatrix} \begin{bmatrix} P_c(1) \\ P_c(\omega_{2n}) \\ P_c(\omega^2_{2n}) \\ \vdots \\ P_c(\omega^{2n-1}_{2n}) \end{bmatrix}$$

   (b) i. To determine the coeffecients of an S degree polynomial P we need to know the value of P at $S+1$ different values. So we pad each polynomial $P_1 ... P_k$ to length $S+1$. Then we take the FFT of all of them in $K * O(SlogS) = O(KSlogS)$ time. Then we multiply all values point wise. Finally we take the IFFT on the product in $O(SlogS)$ time, leading to a final complexity of $O(KSlogS)$.

   ii. Split the polynomials into two groups. Keep splitting each group into two until all groups contain either two or one polynomials. Pad each polynomial with zeroes to length $S+1$. For groups with two polynomials multiply them using the FFT and IFFT in $O(SlogS)$ time. Combine all groups into new groups of two (with a possible group of one). Multiply them as before. Continue until there is only one polynomial. Each level of recursion had at most one multiplication in time $O(SlogS)$ and there were at most $K+1$ levels of recursion leading to a final time complexity of $O((logK)SlogS) = O(SlogSlogK)$.

2. Create the polynomial $V(x) = x^{v_1} + x^{v_2} + ... + x^{v_N}$ with v_i being the value of each of the N coins. Also create the polynomial $R(x) = x^{2v_1} + x^{2v_2} + ... x^{2v_N}$. Pad $V(x)$ to length $2M+1$ with zeroes. The use the FFT and IFFT to compute $F(x) = (V(x))^2$. $V(x)$ is of degree M so this can be done in $O(MlogM)$ time. The powers of $F(x)$ are all possible values of the sums of any two coins in the bag. However $F(x)$ also includes cases of coins being summed with them selves. So we subract $R(x)$ from $F(x)$ to account for the 'no replacement' condition.

3. (a) *Proof.* Base Case $n = 1$: $\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} F_2 & F_1 \\ F_1 & F_0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^1$

   Inductive Hypothesis: Assume the theorem holds true for all values of n up to some k, $k \geq 1$. Inductive Step: Let $n = k+1$. We already have

$$\begin{pmatrix} F_{k+1} & F_k \\ F_k & F_{k-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^k$$

So now we take our RHS

$$\begin{aligned}
\left(\begin{smallmatrix}1&1\\1&0\end{smallmatrix}\right)^{k+1} &= \left(\begin{smallmatrix}1&1\\1&0\end{smallmatrix}\right)^{k}\left(\begin{smallmatrix}1&1\\1&0\end{smallmatrix}\right)\\
&= \begin{pmatrix}F_{k+1} & F_k\\ F_k & F_{k-1}\end{pmatrix}\left(\begin{smallmatrix}1&1\\1&0\end{smallmatrix}\right)\\
&= \begin{pmatrix}F_{k+1}+F_k & F_k+F_{k-1}\\ F_k+F_{k-1} & F_k\end{pmatrix}\\
&= \begin{pmatrix}F_{k+2} & F_{k+1}\\ F_{k+1} & F_k\end{pmatrix}\\
&= \begin{pmatrix}F_{n+1} & F_n\\ F_n & F_{n-1}\end{pmatrix}
\end{aligned}$$

□

(b) At most $2logn$ multiplications are needed, give a time complexity of $O(logn)$. The procedure is outlined as Algorithm 1 at the end of this assignment.

4. We create wishlists for Alice and Bob by rearranging the sequence $[1..N]$ in non-increasing order of how much they are willing to pay for an item. Then we iterate from 1 to N, offering each item to both parties and selling to the highest bidder. Items are not put up for auction if they are already sold and neither Alice nor Bob is offered an item if they have reached their item limit. This method ensures that both parties buy all their most valuable items unless the other is willing to pay more for it. Creating the wish list uses a modified version of Quicksort and runs in $O(nlogn)$, and iterating through $[1..N]$ is linear, leading to $O(nlogn)$ total time complexity. The full algorithm is out lined in Algorithm 2 at the end of this assignment.

5. (a) We iterate through all giants taking the first available leader above height T. We then skip the next K giants and take the next available leader. We continue, keeping track of the number of leaders until we get to the end. We then check that we found at least L leaders, and return True of False depending on that condition.

   (b) We sort H and perform a binary search using the Decision algorithm as our check conditions $logn * O(n) = O(nlogn)$

Algorithm 1 Fibonacci Numbers

function FIBONACCI*(k)
  base ← $\left(\begin{smallmatrix}1&1\\1&0\end{smallmatrix}\right)$
  if $k = 1$ **then**
    return *base*
  else if $k \equiv 1 \bmod 2$ **then**
    return $base*(\text{FIBONACCI}^*(\frac{k-1}{2}))^2$
  else
    return $(\text{FIBONACCI}^*(\frac{k}{2}))^2$
function FIBONACCI(k)
  $M \leftarrow$ FIBONACCI*(k)
  return $M_{1,2}$

Algorithm 2 Bidding War

function GET WISH LIST(S, lo, hi, R)
  $p \leftarrow$PARTITION(S, lo, hi, R)
  GET WISH LIST(S, lo, p, R)
  GET WISH LIST(S, $p+1$, hi, R)
function PARTITION(S, lo, hi, R)
  $pivot \leftarrow S[lo]$
  $j \leftarrow lo - 1$
  $i \leftarrow hi + 1$
  while True **do**
    $i \leftarrow i + 1$
    $j \leftarrow j - 1$
    while $S[i] < pivot$ **do**
      $i \leftarrow i + 1$
    while $S[j] > pivot$ **do**
      $j \leftarrow j - 1$
    if $i \geq j$ **then return** j
    $tmp \leftarrow S[i]$
    $S[i] \leftarrow S[j]$
    $S[j] \leftarrow tmp$
    $tmp \leftarrow R[i]$
    $R[i] \leftarrow R[j]$
    $R[j] \leftarrow tmp$
function ALLOCATE ITEMS(N, A, B, a, b)
  $a_{copy} \leftarrow a$
  $a_{copy} \leftarrow a$
  $aWishList \leftarrow [1..N]$
  $bWishList \leftarrow [1..N]$
  GET WISH LIST(a_{copy}, 1, N, $aWishList$)
  GET WISH LIST(b_{copy}, 1, N, $bWishList$)
  $allocated \leftarrow \{\text{False} | i \in [1..N]\}$
  $aItems \leftarrow 0$
  $bItems \leftarrow 0$
  $sum \leftarrow 0$
  while $i \leq N$ **do**
    if $\neg allocated[aWishList[i]]$ **then**
      if $a[aWishList[i]] \geq b[aWishList[i]] \wedge aItems \leq A$ **then**
        $sum \leftarrow sum + a[aWishList[i]]$
        $aItems \leftarrow aItems + 1$
      else
        $sum \leftarrow sum + b[aWishList[i]]$
        $bItems \leftarrow bItems + 1$
      $allocated[aWishList[i]] \leftarrow \text{True}$
    if $\neg allocated[bWishList[i]]$ **then**
      if $b[aWishList[i]] \geq a[aWishList[i]] \wedge bItems \leq B$ **then**
        $sum \leftarrow sum + b[aWishList[i]]$
        $bItems \leftarrow bItems + 1$
      else
        $sum \leftarrow sum + a[aWishList[i]$
        $aItems \leftarrow aItems + 1$
      $allocated[bWishList[i]] \leftarrow \text{True}$
  return sum

Algorithm 3 Finding Giants

```
function DECISION(N, K, L, H, T)
    leaders ← 0
    gap ← 0
    i ← 0
    while i < N do
        if H[i] ≥ T ∧ gap ≤ 0 then
            leaders ← leaders + 1
            gap ← K
        else
            gap ← gap − 1
        i ← i + 1
    return (leaders ≥ L)
function OPTIMISATION(N, K, L, H)
    H ←QUICKSORT(H)
    top ← N
    bottom ← 0
    max ← 0
    while top ≥ bottom do
        average ← ⌊(top−bottom)/2⌋
        if DECISION(N, K, L, H, average) then
            max ← average
            bottom ← average + 1
        else
            top ← average − 1
    return max
```