ΕΠΛ 436 **Μ. Μαυρονικόλας**

2η Σειρά Ασκήσεων

1. (α) Αφού το διάνυσμα εισόδου έχει 4 στοιχεία, μπορούμε να διαλέξουμε n =4 και $w=e^{2i\pi/4} = e^{i\pi/2}=i$. Υπάρχουν πολλοί τρόποι να υπολογίσουμε τον μετασχηματισμό Fourier του (1,0,0,0). Υπολογίζουμε το $p(x)=1+ox+0x^2+ 0x^3$ sto $(i^0, i^1, i^2, i^3)= (1,i, -1, -i)$. Ή μπορούμε να πολλαπλασιασουμε (1,0,0,0) με τον FFT πίνακα ή να εφαρμόζουμε τον αλγόριθμο διαίρει και βασίλευε. Σε κάθε περίπτωση παίρνουμε FFT((1,0,0,0))=(1,1,1,1).

$$M_4(i) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix} \qquad v = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$M_4(i) \cdot v = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Για να καθορίσουμε το δυάνυσμα για το οποίο ο FFT του ισούται με (1,0,0,0), πολλαπλασίαζουμε με τον αντίστροφο FFT: $M_4(i)^{-1} =(1/4)M_4(-i)$ αφού i-1 = - i

$$M_4(i)^{-1} = \frac{1}{4}M_4(-i) = \frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix} \qquad \frac{1}{4}M_4(-i) \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}$$

Οπότε, ο FFT του (1/4, 1/4,1/4, ¼) είναι το (1,0,0,0).

1(β). Αφού το διάνυσμα εισόδου έχει 4 στοιχεία, μπορούμε να διαλέξουμε n =4 και $w=e^{2i\pi/4} = e^{i\pi/2}=i$. Πάλι, υπάρχουν πολλοί τρόποι να υπολογίσουμε τον μετασχηματισμό Fourier του (1,0,1,-1). Υπολογίζουμε το $p(x)=1+ox+x^2- x^3$ sto $(i^0, i^1, i^2, i^3)= (1,i, -1, -i)$. Ή μπορούμε να πολλαπλασιασουμε (1,0,1,-1) με τον FFT πίνακα ή να εφαρμόζουμε τον αλγόριθμο διαίρει και βασίλευε. Σε κάθε περίπτωση παίρνουμε FFT((1,0,1,-1))=(1,i,3,-i).

$$M_4(i) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix} \qquad v = \begin{bmatrix} 1 \\ 0 \\ 1 \\ -1 \end{bmatrix}$$

$$M_4(i) \cdot v = \begin{bmatrix} 1 \\ i \\ 3 \\ -i \end{bmatrix}$$

Για να καθορίσουμε το δυάνυσμα για το οποίο ο FFT του ισούται με (1,0,1,-1), πολλαπλασίαζουμε με τον αντίστροφο FFT: $M_4(i)^{-1} = (1/4)M_4(-i)$ αφού i-1 = - i .

$$M_4(i)^{-1} = \frac{1}{4}M_4(-i) = \frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix} \qquad \frac{1}{4}M_4(-i) \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1/4 \\ -i/4 \\ 3/4 \\ i/4 \end{bmatrix}$$

Οπότε, ο FFT του (1/4, (-1/4)i, 3/4, (¼) i) είναι το (1,0,1,-1).

2(a). Έστω p(x)=x+ 1 , q(x) = x^2+1 Αφού ο βαθμός του p(x)q(x) είναι 3, μπορούμε να υπολογίσουμε τον FFT χρησιμοποιώντας την 4η ρίζα της μονάδας, που είναι $w=e^{2i\pi/4} = e^{i\pi/2}$=i. Πάλι, υπάρχουν πολλοί τρόποι να υπολογίσουμε τον μετασχηματισμό Fourier του p(x)q(x). Μπορούμε να υπολογίζουμε το p(x) (αντίσ. το q(x)) στο (i^0, i^1, i^2, i^3)= (1,i, -1, -i). Ή να πολλαπλασιάσουμε τους συντελεστές του p(x) ((αντίσ. του q(x)) που είναι το (1,1,0,0) με τον FFT πίνακα. ή να εφαρμόζουμε τον αλγόριθμο διαίρει και βασίλευε. Σε κάθε περίπτωση παίρνουμε FFT(p(x)) = (2,1+i,0, 1-i) και FFT(q(x)) = (2,0,2, 0).

$$M_4(i) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix} \qquad P = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \qquad Q = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

$$M_4(i) \cdot P = \begin{bmatrix} 2 \\ 1+i \\ 0 \\ 1-i \end{bmatrix} \qquad M_4(i) \cdot Q = \begin{bmatrix} 2 \\ 0 \\ 2 \\ 0 \end{bmatrix}$$

|

Πολλαπλασιάζοντας τα components παίρνουμε FFT(p(x)q(x)) =(4, 0 ,0, 0). Για να υπολογίσουμε τους συντελεστές του p(x)q(x) χρησιμοποιούμε τον αντίστροφο FFT. p(x)q(x) = FFT^{-1} (FFT (p(x)q(x))) = FFT^{-1} ((4, 0, 0,0)). Ο FFT^{-1} υλοποιείται πολλαπλασιάζοντας το (4, 0, 0,0) με το $M_4(i)^{-1} = (1/4)M_4(-i)$ αφού $i^{-1}= -i$.

$$M_4(i)^{-1} = \frac{1}{4}M_4(-i)\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix} \qquad \frac{1}{4}M_4(-i)\cdot\begin{bmatrix} 4 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Έτσι βρίσκουμε το πολυώνυμο $p(x)q(x)= x^3+ x^2+ x +1$.

2(b). Έστω $p(x)=2x+x+ 1$, $q(x) = 3x +2$. Αφού ο βαθμός του p(x)q(x) είναι 3, μπορούμε να υπολογίσουμε τον FFT χρησιμοποιώντας την $4^η$ ρίζα της μονάδας, που είναι $w=e^{2iπ/4} = e^{iπ/2}=i$. Πάλι, χρησιμοποιώντας ένα από τους τρόπους που αναφέρθηκαν στο μερος (α) μπορούμε να υπολογίσουμε FFT(p(x)) = (4, -1+i, 2, -1-i) και FFT(q(x)) = (5,2+3i.,-1, 2-3i).

$$M_4(i) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix} \qquad P = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 0 \end{bmatrix} \qquad Q = \begin{bmatrix} 2 \\ 3 \\ 0 \\ 0 \end{bmatrix}$$

$$M_4(i)\cdot P = \begin{bmatrix} 4 \\ -1+i \\ 2 \\ -1-i \end{bmatrix} \qquad M_4(i)\cdot Q = \begin{bmatrix} 5 \\ 2+3i \\ -1 \\ 2-3i \end{bmatrix}$$

Πολλαπλασιάζοντας τα components παίρνουμε FFT(p(x)q(x)) =(20, -5-i ,-2, -5+i). Για να υπολογίσουμε τους συντελεστές του p(x)q(x) χρησιμοποιούμε τον αντίστροφο FFT. p(x)q(x) = FFT^{-1} (FFT (p(x)q(x))) = FFT^{-1} ((20, -5-i, -2,-5+i)). Ο FFT^{-1} υλοποιείται πολλαπλασιάζοντας το (20, -5-i, -2,-5+i) με το $M_4(i)^{-1} = (1/4)M_4(-i)$ αφού $i^{-1}= -i$.

$$M_4(i)^{-1} = \frac{1}{4}M_4(-i)\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix} \qquad \frac{1}{4}M_4(-i)\cdot\begin{bmatrix} 20 \\ -5-i \\ -2 \\ -5+i \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \\ 7 \\ 6 \end{bmatrix}$$

3.

Algorithm 1: kSmallest

Input: $A[1, \cdots, m], B[1, \cdots, n], m, n, k$; A and B are two sorted lists, it is sufficient to assume $m, n \leq k$

Output: $C[1, \cdots, k]$;The kth smallest numbers of the union of A and B

1 **if** $m = 1$ *or* $n = 1$ **then return** (the k smallest elements in constant time);
2 **if** $A[\lceil \frac{m}{2} \rceil] < B[\lceil \frac{n}{2} \rceil]$ **then**
3 **if** $k \geq \lceil \frac{m}{2} \rceil + \lceil \frac{n}{2} \rceil$ **then**
4 **return** $kSmallest(A[1 + \lceil \frac{m}{2} \rceil, \cdots, m], B[1, \cdots, n], \lfloor \frac{m}{2} \rfloor, n, k - \lceil \frac{m}{2} \rceil)$
5 **else**
6 **return** $kSmallest(A[1, \cdots, m], B[1, \cdots, \lceil \frac{n}{2} \rceil], m, \lceil \frac{n}{2} \rceil, k)$
7 **end**
8 **else**
9 **if** $k \geq \lceil \frac{m}{2} \rceil + \lceil \frac{n}{2} \rceil$ **then**
10 **return** $kSmallest(A[1, \cdots, m, B[1 + \lceil \frac{n}{2} \rceil, \cdots, n], m, \lfloor \frac{n}{2} \rfloor, k - \lceil \frac{n}{2} \rceil)$
11 **else**
12 **return** $kSmallest(A[1, \cdots, \lceil \frac{m}{2} \rceil], B[1, \cdots, n], \lceil \frac{m}{2} \rceil, n, k)$
13 **end**
14 **end**
15

Ανάλυση ορθότητας: Θεωρούμε ότι m, n≤k, αφού είναι αρκετό να δούμε μόνο τα πρώτα k στοιχεία των A, B ψάχνοντας για το k στο μικρότερο στοιχείο της ένωσης των A, B. Ο αλγόριθμος ξεκινά συγκρίνοντας τα A[⌈m/2⌉] και B[⌈n/2⌉].
Σημειώστε ότι για να είναι ένας αριθμός ένας από τους k μικρότερους αριθμός σε μια λίστα, μπορεί να είναι μεγαλύτερος από το πολύ k-1 αριθμούς στη λίστα.

Υποθέστε ότι A[⌈m/2⌉] < B[⌈n/2⌉] για κάποιο στοιχείο στο B[⌈n/2⌉ +1, …, n], είναι μεγαλύτερο από τουλάχιστον (m+n)/2 της ένωσης των A, B. Θεωρείστε τις δύο παρακάτω περιπτώσεις:
(1). (m+n)/2≥k, αφού κάθε στοιχείο στο στο B[⌈n/2⌉ +1, …, n], είναι μεγαλύτερο από τουλάχιστον (m+n)/2 >k αριθμούς, δεν μπορεί να είναι το k-στο μικρότερο στοιχείο στη λίστα. Αφαιρούμε τους αριθμού αυτούς και ψάχνουμε για τον k-στο μικρότερο στοιχείο από το A[1+⌈m/2⌉, …, m] , B[1, …,⌈n/2⌉].
(2) (m+n)/2<k, αφού κάθε στοιχείο στο A[1+⌈m/2⌉, είναι μικρότερο από τουλάχιστον (m+n)/2 αριθμούς, είναι μεγαλύτερο από το πολύ k-1 αριθμούς στη λίστα. Οπότε όλα τα στοιχεία A[1+⌈m/2⌉ πρέπει να είναι στο σύνολο των το k-στο μικρότερων στοιχε΄λιων της ένωσης. Το μόνο που χρειάζεται να κάνουμε έιανι να βρούμε τα υπολοιπα (k-m/2) στοιχεία από τα A[1+⌈m/2⌉, …, m] , B[1, …,⌈n⌉].
Λόγω συμμετρίας μεταξύ των A, B η άλλη περίπτωση A[⌈m/2⌉] ≥ B[⌈n/2⌉] , αναλύεται παρόμοια.

Ανάλυση χρόνου: Σε κάθε βήμα του αλγόριθμου για την μείωση της λιστας θα χρειαστούμε O(log(m)+log(n)) βήματα. Σε κάθε βήμα κάνουμε δουλειά σταθερού χρόνου. Οπότε συνολικά έχουμε χρόνο O(log(m)+log(n)).

4 (α).
Algorithm (4 points):

Algorithm 2: CHECKCOUNT

Input: a, $A[1, \cdots, n]$
Output: The number of occurrences of a in $A[1, \cdots, n]$

```
1 k = 0;
2 for each A[i] do
3 |   if A[i] = a then
4 |   |   k = k + 1;
5 |
6 end
7 return(k)
```

Algorithm 3: MAJORITY1

Input: $A[1, \cdots, n]$
Output: The majority of $A[1, \cdots, n]$ if it exists

```
1 if n = 1 then return (A(1))
2 if exists a = Majority(A[1, ..., ⌈n/2⌉]) then
3 |   if checkcount(a, A[1, ..., n)) > n/2 then return (a)
4 else if exists a = Majority(A⌈n/2⌉ + 1, ..., n) then
5 |   if checkcount(a, A[1, ..., n)) > n/2 then return (a)
6 end
7 return ("No majority element.")
```

Ανάλυση ορθότητας: εάν το Α έχει ένα στοιχείο πλειοψηφίας υ και Α1 , Α2 είναι τα δύο μισά του Α, τότε το υ πρέπει να είναι επίσης ενα στοιχείο πλειοψηφίας στο Α1 ή το Α2 ή και στα δύο. Εναλλακτικά, αν το υ δεν είναι στοιχείο πλειοψηφίας ούτε το Α1 ούτε στο Α2, τότε το υ δεν μπορεί να είναι στοιχείο πλειοψηφίας στο Α. Ο αλγόριθμος αναδρομικά υπολογίζει τη πληοψηφία στα δύο μισά και ελέγχει αν είναι επίσης το στοιχείο πλειοψηφίας πριν τον διαχωρισμό.

Ανάλυση χρόνου: Ο αλγόριθμος αναδρομικα διαιρεί το πρόβλημα σε 2 υποπροβλήματα μεγέθους n/2 και ελέγχει εάν το στοιχείο πλειοψηφίας στο διαχωρισμένο πίνακα είναι επίσης στοιχείο πλειοψηφίας στον μη διασπασμένο, παίρνει γραμμικό χρόνο.Έτσι ο συνολικός χρόνος είναι T(n)=2T(n/2)+O(n)= O(nlog n).

4 (b).

.

Algorithm 4: MAJORITY2

```
Input: A[1, ··· , n]
Output: The majority of A[1, ··· , n] if it exists
1  b = GetCandidate(A[1, ··· , n])
2  if checkCount(b, A[1, ··· , n]) > n/2 then return b
3  else
4  |  return "No majority element."
5  end
```

Algorithm 5: GETCANDIDATE

```
Input: A[1, ··· , n]
Output: A possible majority (candidate solution) of A[1, ··· , n] if it
        exists (may return false positive)
1  if  n = 0 then return 0 if  n =1 then return (A(1))
2  APair[1, ··· , n/2][2] ← {A[1, ··· , n]} ; pair up the elements in A
   randomly
3  k = 0;
4  for each APair[i] do
5  |  if APair[i][1] = APair[i][2] then
6  |  |  A[k] = APair[i][1];
7  |  |  k = k + 1;
8  |
9  end
10 return(GetCandidate(A[1, ··· , k])
```

Ανάλυση ορθότητας:
Υποθέτουμε ότι το μέγεθος του αρχικού πινακα είναι δύναμη του 2. Πρέπει να αποδείξουμε ότι η πλειοψηφία στο A_k είναι επίσης πληοψηφία στο A_{k+1} , όπου το A_{k+1} είναι ένας πίνακας που προκύπτει μετά την εφαρμογή της pair-and-delete διαδικασίας στο A_k, όπως περιγράφεται από τον αλγόριθμο.
Υποθέστε ότι το A_k έχει μέγεθος n και στοιχείο πλειοψηφίας b, θέτωντας n_b ίσο με τον αριθμό των εμφανίσεων του b. Οπότε
$n_b > n/2$. Θέτωντας n_x ο αριθμός των b-x ζευγαριών που σχηματίζονται στη γραμμή 5, όπου x είναι οποιοδήποτε στοιχείο διαφοερτικό του b. Αυτό αφήνει $(½)(n_b - n_x)$ b-b ζευγάρια. Οπότε το πλήθος των b που παραμένουν στο A_{k+1} είναι $n_b - n_x - (½)(n_b - n_x) = (½)(n_b - n_x)$.
Το συνολικό μέγεθος του A_{k+1} είναι μεγαλύτερο όταν τα λιγότερα στοιχεία έχουν αφαιρεθεί. Έχουμε ως τώρα ζευγαρώσει $n_b + n_x$ στοιχεία , από τα οποία έχουμε αφαιρέσει $2n_x - (½)(n_b - n_x)$ στοιχεία. Από τα υπόλοιπα n- $(n_b + n_x)$ στοιχεία, πρέπει να αφαιρέσουμε τουλάχιστον τα μισά από αυτά. Οπότε συνολικά θα αφαιρέσουμε τουλάχιστον $(n_b + n_x)$ στοιχεία από το A_{k+1} , δίνοντας το πολύ $(½)n - n_x$ στοιχεία στο A_{k+1}.

Έτσι το κλάσμα των b's στο A_{k+1} είναι τουλάχιστον $((½)n_b - n_x/2)/(n/2 - n_x) > (n/4 - n_x/2)/(n/2 - n_x) = 1/2$, αφού $n_b > n/2$. Αυτό σημαίνει ότι η πλειοψηφία στο A_{k+1} είναι το b.

Οπότε ο αλγόριθμος είναι σωστός όταν η πλειοψηφία στον πίνακα είναι το b. Όμως, εάν το αρχικό πίνακας δεν έχει στοιχείο πλειοψηφίας, ο αλγόριθμος μπορεί να παράξει ένα λανθασμένο ακαίρεο, έτσι θα πρέπει να ελέξουμε ότι το b είναι πράγματι η πλειοψηφία στον αρχικό πίνακα.

Ανάλυση χρόνου: Ο αλγόριθμος ζευγαρώνει τα στοιχεία του A και το πολύ ένα είναι αριστερά για κάθε ζευγάρι σύγκρισης (O(n)) και ανάγουμε σε ένα υποπρόβλημα μεγέθους το πολύ n/2. Οπότε ο χρόνος του αλγόριθμου είναι T(n)=T(n/2)+O(n)=O(n).

6. Έστω S(n) ο χρόνος για να τετραγωνίσουμε n-bit ακαιρέους, M(n) ο χρόνος για να πολλαπλασιάσουμε δυο n-bit ακαιρέους. Μπορούμε να πολλαπλασίασουμε n-bit ακαιρέους a , b κάνοντας 3 τετράγωνα, 3 προσθέσεις και 1 ένα-bit right-shifting χρησιμοποιώντας ab =((a+b)2 –a2 –b2 >> 1, M(n) = 3S(n) +O(n).

S(n)=Ω(n), τότε M(n)=Θ(S(n)). Αυτό σημαίνει ότι ο πολλαπλασιασμός δυο n-bit ακαιρέων μπορεί να γίνει ασυμπτοτικά όσο χρήγορα όσο θέλουμε για να τετραγωνίσουμε ένα n-bit ακαιρέο, που έρχεται σε αντίφαση με το Professor Lake claim.