**ממן 15 – מבני נתונים ומבוא לאלגוריתמים – ברנדס איתי ת.ז.**

**שאלה 1:**

1. נשתמש במודל עץ ההחלטה שהוגדר בעמוד 138 בספר הלימוד (תחת הנושא "מודל עץ ההחלטה") ע"מ להוכיח שכל אלגוריתם מיון מבוסס השוואות הממיין מערך באורך 5 חייב לבצע במקרה הגרוע 7 השוואות לפחות.

כמצוין בספר הלימוד, אורכו של המסלול הארוך ביותר מהשורש של עץ ההחלטה לאחד העלים הנגישים בעץ מייצג את מספר ההשוואות שאלגוריתם המיון עורך במקרה הגרוע.

נסמן את אורך המסלול הארוך ביותר לעיל ב-h, שזהו גם מספר ההשוואות שהאלגוריתם עורך במקרה הגרוע.  
את מספר העלים בעץ ההחלטה נסמן ב-l.

עלי עץ ההחלטה של כל אלגוריתם מיון מבוסס-השוואה מכילים את כל תמורות הקלט, ז"א במקרה שלנו, קיימים לפחות עלים.

בנוסף, עץ בינארי בגובה h מכיל לכל היותר עלים, ולכן:

ניקח לוגריתם של שני האגפים, ונקבל, משום שפונקציית הלוגריתם הינה פונקציה מונוטונית עולה:

*ונקבל ש- (שכן גובה העץ הוא ודאי מספר טבעי).*

*ולכן,* כל אלגוריתם מיון מבוסס השוואות הממיין מערך באורך 5 חייב לבצע במקרה הגרוע 7 השוואות לפחות.

1. השגרה FIVE-ELEMENTS-SORT מקבלת מערך באורך 5 וממיינת אותו בעזרת 7 השוואות.

אנו מניחים שקיימת שגרת עזר CREATE-ARRAY המקבלת 5 משתנים ומכניסה אותם למערך חדש באורך 5.

FIVE-ELEMENTS-SORT(A)

► First, we give the elements new names: a,b,c,d,e.

1 a 🡨A[1]; b 🡨A[2]; c 🡨A[3]; d 🡨A[4]; e 🡨A[5]

► Then we sort pairs (a,b) and (c,d)

2 if a > b

3 then SWAP a <-> b

4 if c > d

5 then SWAP c <-> d

► Now a b, c d.

► Then we sort (b,d)

6 if b>d

7 then SWAP b <-> d

8 SWAP a <-> c

► Now a bd, c d.

► Then, we sort subarray [a,b,d,e] and put the result in array B

9 if e<b

10 if e<a

11 then B 🡨CREATE-ARRAY(e,a,b,d,NIL)

12 d\_new\_index 🡨 4

13 else

14 then B 🡨CREATE-ARRAY(a,e,b,d,NIL)

15 d\_new\_index 🡨 4

16 elif e<d

17 then B 🡨CREATE-ARRAY(a,b,e,d,NIL)

18 d\_new\_index 🡨 4

19 else

20 then B 🡨CREATE-ARRAY(a,b,d,e,NIL)

21 d\_new\_index 🡨 3

► Now the first four elements in B are sorted: .

► We also know that c<d=B[d\_new\_index].

► Then, we put the element c, in the right place in the array.

►c<d=B[d\_new\_index], and d\_new\_index is 3 or 4, so c can be in indexes 1,2,3,4

►only.

22 if c < B[1]

► Order must be c,B[1],B[2],B[3],B[4]

23 then A 🡨CREATE-ARRAY(c,B[1],B[2],B[3],B[4])

24 elif c < B[2]

► Order must be B[1],c,B[2],B[3],B[4]

25 then A 🡨CREATE-ARRAY(B[1],c,B[2],B[3],B[4])

26 elif d\_new\_index == 3

► Order must be B[1],B[2],c,B[3],B[4], since c<d=B[3]

27 then A 🡨CREATE-ARRAY(B[1],B[2],c,B[3],B[4])

28 else

► Order must be B[1],B[2],B[3],c,B[4], since c<d=B[4]

29 then A 🡨CREATE-ARRAY(B[1],B[2],B[3],c,B[4])

30 return A

ההסברים לנכונות השגרה ניתנים בתיעוד השגרה עצמה.

נראה שאכן האלגוריתם משתמש בבדיוק 7 השוואות.

בשורות 2-5 האלגוריתם משתמש ב-2 השוואות.

בשורות 6-8 ישנו שימוש בעוד השוואה אחת.

בשורות 9-21:

* אם התנאי בשורה 9 מתקיים, אז משתמשים ב2 השוואות: בשורה 9 ובשורה 10.
* אם התנאי בשורה 9 לא מתקיים, אז משתמשים ב2 השוואות גם כן: בשורה 9 ובשורה 16.

לסיכום, גם בשורות 9-21 אנו משתמשים בדיוק ב2 השוואות.

בשורות 22-30 אנו משתמשים בעוד 2 השוואות במקרה הגרוע – בשורות 22 ו-24 (ההשוואה בשורה 26 לא מתבצעת אל מול תא במערך אלא מול משתנה מחוץ למערך, ולכן לא נחשבת השוואה).

סה"כ האלגוריתם משתמש ב-7 השוואות בדיוק.

1. נבחן כיצד אלגוריתמי המיון מבוססי ההשוואות שלמדנו בקורס מתנהגים עבור מערך באורך 5, ונראה האם ביניהם קיים אלגוריתם שמקיים 7 השוואות במקרה הגרוע:
2. מיון בועות (עמוד 18 במדריך הלמידה, תחת הפרק "מיון-בועות")

* הלולאה החיצונית בשורה 1 רצה 5 פעמים, והלולאה הפנימית בשורה 2 תרוץ 4,3,2,1,0 פעמים (בהתאמה למספר האיטרציה בלולאה החיצונית), כך שסה"כ שורות 3-4, שמבצעות השוואה אחת, תרוץ פעמים.
* לסיכום, מיון בועות מבצע **לפחות 10 השוואות** במקרה הגרוע, ואיננו עומד בתנאי המבוקש.

1. מיון הכנסה (עמוד 14 בספר הלימוד, בתחתית העמוד)

* נבחר מערך ממוין יורד. הלולאה החיצונית בשורה 1 רצה 4 פעמים, והלולאה הפנימית בשורה 5 תרוץ 1,2,3,4 פעמים (בהתאמה למספר האיטרציה בלולאה החיצונית), כך שסה"כ שורה 5, שמבצעת השוואה אחת, תרוץ פעמים.
* לסיכום, מיון הכנסה מבצע **לפחות 10 השוואות** במקרה הגרוע, ואיננו עומד בתנאי המבוקש.

1. מיון בחירה (עמוד 13 במדריך הלמידה, תחת הפרק "האלגוריתם מיון-בחירה")

* נבחר מערך ממוין יורד. הלולאה החיצונית בשורה 1 רצה 4 פעמים, והלולאה הפנימית בשורה 3 תרוץ 1,2,3,4 פעמים (בהתאמה למספר האיטרציה בלולאה החיצונית), כך שסה"כ שורה 4, שמבצעת השוואה אחת, תרוץ פעמים.
* לסיכום, מיון בחירה מבצע **לפחות 10 השוואות** במקרה הגרוע, ואיננו עומד בתנאי המבוקש.

1. מיון מיזוג (עמוד 25 בספר הלימוד, שגרת MERGE, ועמוד 27 בספר הלימוד, שגרת MERGE-SORT)

* נבחר מערך ממוין יורד [5,4,3,2,1]. המערך מחולק בשגרה ל2 תתי-מערכים [5,4,3] ו-[2,1].
* נבחין שהשגרה MERGE על תתי-מערכים באורך 1 לא מבצעת השוואות בין 2 איברי המערך. אינני בטוח האם השוואה כנגד זקיף (שגרת Merge שורות 8-9) נחשבת ל"השוואה" כהגדרתה בתרגיל, אך נניח שהיא איננה נחשבת, שכן ניתן להפריך את הטענה גם מבלי להתייחס להשוואות אל מול הזקיף.
  + השגרה MERGE-SORT מחלקת את המערך [5,4,3] ל2 תתי-מערכים [5,4] ו-[3].
    - השגרה MERGE מבצעת על התת-מערכים [5],[4] 2 ריצות של הלולאה בשורה 12, ולכן מתרחשות כביכול 2 השוואות בשורה 13, אך אחת מהן היא השוואה כנגד ערך זקיף (משורות 8-9) ולכן מתבצעת **השוואה** **אחת**.
    - השגרה MERGE מבצעת על התת-מערכים [4,5],[3] 3 ריצות של הלולאה בשורה 12, ולכן מתרחשות כביכול 3 השוואות בשורה 13, אך אחת מהן היא השוואה כנגד ערך זקיף ולכן מתבצעות **שתי השוואות**.
  + השגרה MERGE מבצעת על התת-מערכים [2],[1] 2 ריצות של הלולאה בשורה 12, ולכן מתרחשות כביכול 2 השוואות בשורה 13, אך אחת מהן היא השוואה כנגד ערך זקיף ולכן מתבצעת **השוואה** **אחת**.
  + השגרה MERGE מבצעת על התת-מערכים [3,4,5],[1,2] 5 ריצות של הלולאה בשורה 12, ולכן מתרחשות כביכול 5 השוואות בשורה 13, אך אחת מהן היא השוואה כנגד ערך זקיף (משורות 8-9) ולכן מתבצעות **4 השוואות** במקרה הגרוע.
* לסיכום, מיון מיזוג מבצע **לפחות 8 השוואות** במקרה הגרוע, ואיננו עומד בתנאי המבוקש.

1. מיון מהיר (עמוד 122 בספר הלימוד, שגרות QUICKSORT וPARTITION)

* נבחין שהשגרה PARTITION על תתי-מערכים באורך 1 לא מבצעת השוואות כלל.
* נבחר מערך ממוין יורד [5,4,3,2,1]. בשורה 2 נקרא לשגרה PARTITION עם מערך זה, שבה יש לולאה פנימית בשורה 3 אשר רצה 4 פעמים, ובכל פעם מבצעת השוואה בשורה 4, לכן מתבצעות **4 השוואות**.
* מתרחשת קריאה לשגרה PARTITION עם התת-מערך [5,4,3,2], ומאותו טיעון שהוסבר בנקודה הקודמת, מתבצעות **3 השוואות**.
* מתרחשת קריאה לשגרה PARTITION עם התת-מערך [5,4,3], ומאותו טיעון שהוסבר בנקודה הקודמת, מתבצעות **2 השוואות**.
* כבר עכשיו אנו מבצעים **לפחות 9 השוואות** במקרה הגרוע, לכן מיון מהיר איננו עומד בתנאי המבוקש.

1. מיון ערמה (עמוד 113 בספר הלימוד, שגרת HEAPSORT תחת הנושא "האלגוריתם מיון-ערמה", אשר משתמשת בשגרות BUILD-MAX-HEAP מראשית עמוד 111 וב-MAX-HEAPIFY מראשית עמוד 109)

* נבחר מערך ממוין עולה [1,2,3,4,5].
* נבחין שבקריאה של MAX-HEAPIFY על עלים לא מתבצעות שום השוואות.
* בשגרה HEAPSORT בשורה 4 אנו קוראים לBUILD-MAX-HEAP, שהוא קורא ל-MAX-HEAPIFY בשורה 3 עם i=2. השגרה MAX-HEAPIFY מבצעת **2 השוואות**: אחת בשורה 3 ואחת בשורה 6.
* כעת מתבצעת קריאה ל-MAX-HEAPIFY בשורה 3 עם i=1. השגרה MAX-HEAPIFY מבצעת **2 השוואות**: אחת בשורה 3 ואחת בשורה 6.
* השגרה קוראת רקורסיבית ל MAX-HEAPIFYעם i=2, שכבר ראינו שמבצע **2 השוואות**.
* עד כה האלגוריתם ביצע 6 השוואות.
* כעת השגרה BUILD-MAX-HEAP מסיימת את עבודתה, ומתקבלת ערמה בעלת הייצוג המערכי הבא: [1,5,3,4,2].
* הלולאה בשורה 2 בHEAPSORT רצה 4 פעמים, ובכל פעם קוראת ל-MAX-HEAPIFY עם i=1. כבר ראינו שקריאה לMAX-HEAPIFY עם i=1 פירושה לפחות 2 השוואות, לכן האלגוריתם מבצע עוד לפחות 8 השוואות.
* לסיכום, מיון ערמה מבצע **לפחות 14 השוואות** במקרה הגרוע, ואיננו עומד בתנאי המבוקש.

הראנו שלכל אלגוריתם מן האלגוריתמים מבוססי ההשוואות שלמדנו בקורס קיים מערך מאורך 5 עבורו המיון מתבצע בעזרת 8 או יותר השוואות, ולכן אף אחד מאלגוריתמים אלה לא מקיים את התנאי הנדרש בסעיף ב'.

**שאלה 2:**

בפתרון נשתמש בשגרת MERGE-SORT מעמוד 27 בספר הלימוד, ובשגרת RADIX-SORT מעמוד 143 בספר הלימוד, כאשר נבחר את המיון הפנימי בRADIX-SORT להיות מיון מניה, מעמוד 140 בספר הלימוד.

IS\_Z\_SUM\_OF\_TWO\_DIFF\_VALUES(S, k, z)

► First, we sort the array.

1 if

► If , we’ll use merge sort, since it has better complexity.

2 then MERGESORT(S, 1, length[S])

3 else

► Otherwise, we’ll use radix sort, since it has better complexity in

this case. We know that each element is a native number between 0 and

, so we can look at each element as a number of k digits in base n.

The internal sorting procedure is counting sort, which is stable, therefore the

Radix Sort will work correctly.

4 then RADIX-SORT(S, k)

► Now we search for two different values in the array that sum up to z.

i will be pointing to the beginning of the array, and j to the end.

i will increase by one, and j will decrease by one, until we’ve checked all the relevant

pairs of elements in the array that might sum up to k.

5 i 🡨 1

6 j 🡨 length[S]

7 while (i<j)

8 do if S[i]+S[j] > k

► If S[i]+S[j] > k, increasing i by one will only make the sum larger, since the array’s sorted and A[i] <= A[i+1]. Therefore, we decrease j by one, and since A[j-1] <= A[j], the new sum might add up to exactly z.

9 then j 🡨 j-1

10 elif S[i]+S[j] < k

► If S[i]+S[j] < k, decreasing j by one will only make the sum smaller, since the array’s sorted and A[j-1] <= A[j]. Therefore, we increase i by one, and since A[i] <= A[i+1], the new sum might add up to exactly z.

11 then i 🡨 i+1

12 else

► Then, S[i] + S[j] == k, it’s a hit.

13 return TRUE

14 return FALSE

ההסברים לנכונות השגרה ניתנים בתיעוד השגרה עצמה.

ננתח את סיבוכיות השגרה:

ראשית, בשורות 1-4 אנו ממיינים את המערך. אנו משתמשים במיון מיזוג או מיון בסיס – בהתאם למיון שיקנה לנו סיבוכיות טובה יותר לריצה.

אם , שורה 2 רצה, ואנו ממיינים בעזרת מיון מיזוג בזמן ריצה של (כמוכח בעמוד 29 בספר הלימוד).

אחרת, שורה 4 רצה, ונמיין בעזרת מיון בסיס. בשורה 1 בשגרת RADIX-SORT (ע"מ 143 בספר הלימוד) אנו רצים בלולאה על k איברים – מספר הספרות במספרים. בשורה 2 בשגרת RADIX-SORT אנו קוראים למיון הפנימי, מיון ספירה, על n ספרות. מיון ספירה ממיין בזמן ריצה של , (כמוכח בעמוד 141 בספר הלימוד), ולכן שורה 4 תרוץ בזמן של .

לסיכום, זמן הריצה של שורות 1-4 הוא .

הלולאה בשורות 7-13 רצה עד ש-. בשורות 5-6 אנו מאתחלים את i,j להצביע על תחילת וסוף המערך, ובכל איטרציה של הלולאה אנו מקדמים את i ב1 (שורה 11), מפחיתים את j ב1 (שורה 9), או יוצאים מן הלולאה (שורה 13).

במקרה הגרוע, הלולאה בשורות 7-13 תרוץ עד ש , ואז פירוש הדבר הוא שהלולאה רצה n-1 פעמים. כל איטרציה רצה בסיבוכיות קבועה. לפיכך, הלולאה בשורות 7-13 רצות בזמן של .

לסיכום, השגרה רצה ב, שזהו כמובן .

1. *(ההבדלים משגרה בסעיף א' מסומנים במרקר צהוב. תיעודי קוד זהים נשמטו בפתרון זה)*

IS\_Z\_SUM\_OF\_THREE\_DIFF\_VALUES(S, k, z)

1 if

2 then MERGESORT(S, 1, length[S])

3 else

4 then RADIX-SORT(S, k)

5 for x🡨1 to length[S]-2

6 then i 🡨 x+1

7 j 🡨 length[S]

8 while (i<j)

9 do if S[i]+S[j]+S[x] > k

10 then j 🡨 j-1

11 elif S[i]+S[j]+S[x] < k

12 then i 🡨 i+1

13 else

14 return TRUE

15 return FALSE

נסביר בקצרה את נכונות השגרה:

רוב הקוד מועתק מסעיף א', שאת נכונותו כבר הוכחנו.

ההבדל הוא שבמקום לבדוק האם 2 איברים סכומם k, תוך כדי הזזת האינדקסים i,j אחד לקראת השני, כעת אנו רצים בלולאה שבשורה 5, מקבעים אינדקס x כלשהו, ואז נותנים לאינדקסים i,j לרוץ כמו שעשינו בסעיף א', תוך כדי בדיקה האם S[i]+S[j]+S[x]==k. בכל איטרציה אנו מקדמים את ערך ה-x ב-1, החל מהאיבר הראשון ועד length[S]-2 (שכן במקרה זה i ו-j מקבלים את 2 האיברים הנותרים), כך שבסופו של דבר נריץ את אלגוריתם החיפוש שבסעיף א' על איברי המערך שעלולים לקיים את התנאי S[i]+S[j]+S[x]==k.

ננתח את סיבוכיות השגרה:

כפי שהוכחנו בסעיף א', זמן הריצה של שורות 1-4 הוא .

זמן הריצה של שורות 6-14 הן, כפי שהוכחנו בסעיף א', (ברור שהתוספת של חיבור S[x] להשוואות בשורות 9 ו-11 מוסיפות זמן ריצה קבוע בלבד).

הלולאה בשורות 5-14 רצה length[S]-2 פעמים, ולכן זמן הריצה של הלולאה הוא .

לסיכום, זמן הריצה של השגרה כולה הוא .

**שאלה 3:**

בתרגיל נשתמש בשגרות הבסיסיות של מחסנית, כמוגדרות בספר הלימוד בעמוד 169.

בפתרון גם נניח שקיימת שגרה CREATE-STACK היוצרת מחסנית חדשה בגודל N.

Q3-STACK-PUSH-EXTEND-SINGLE(S, x)

► if the latest index in the stack is the size of the array, the stack is full and we need to extend it.

1 if top[S] == length[S]

► we’ll create a temp stack of size N

2 then T 🡨 CREATE-STACK(length[S])

► Copy all values to the temp stack. The values will be stored in a reversed order compared to the original stack.

3 while not STACK-EMPTY(S)

4 do PUSH(T, POP(S))

► Create a new stack of size N+1

5 S 🡨 CREATE-STACK(length[S]+1)

► Copy all values from the temp stack to the new stack. Values will be reversed again, so they’ll be the same as the original stack, just that now the array length is larger by 1 element.

6 while not STACK-EMPTY(T)

7 do PUSH(S, POP(T))

8 PUSH(S, x)

9 return S

ההסברים לנכונות השגרה ניתנים בתיעוד השגרה עצמה.

ננתח את זמן הריצה עבור סדרה של n פעולות הכנסה החל ממחסנית ריקה:

כמצוין בעמוד 169 בספר הלימוד, כל אחת מהפעולות STACK-EMPTY, PUSH, POP מתבצעת בזמן קבוע.

משום שהמחסנית ריקה בהתחלה, ואנו מבצעים n פעולות הכנסה, בכל פעולה כזו יתקיים תנאי 1, ולכן השורות 2-7 ירוצו n פעמים.

עבור פעולה מספר i, שורות 2,5 יוצרות מחסניות ריקות חדשות בזמן ריצה לינארי (לפי i),

והלולאות בשורות 3,6 רצות length[T]=i פעמים, כמספר איברי המערך, ובכל פעם מבצעות מספר פעולות קבוע.

ז"א בסה"כ בתוכנית, שורות 2-7 ירוצו יחד פעמים, כמובן, בסיבוכיות של .(k זהו הקבוע שב-)

שורה 8 רצה בסיבוכיות קבועה.

ולכן מתקיים:

1. נניח שלמחסנית S קיים מאפיין extendby[S] המחזיר את מספר התאים שבה גדלה המחסנית בעת פעולת הרחבה.

(ההבדלים משגרת סעיף א' מסומנים במרקר צהוב. תיעודי קוד זהים נשמטו בפתרון זה.)

Q3-STACK-PUSH-EXTEND-MULTI(S, x)

1 if top[S] == length[S]

2 then T 🡨 CREATE-STACK(length[S])

3 while not STACK-EMPTY(S)

4 do PUSH(T, POP(S))

► Create a new stack of size N+extendby[S]

5 S 🡨 CREATE-STACK(length[S]+extendby[S])

6 while not STACK-EMPTY(T)

7 do PUSH(S, POP(T))

8 PUSH(S, x)

9 return S

ננתח את זמן הריצה עבור סדרה של n פעולות הכנסה החל ממחסנית ריקה:

כמצוין בעמוד 169 בספר הלימוד, כל אחת מהפעולות STACK-EMPTY, PUSH, POP מתבצעת בזמן קבוע.

תנאי 1 בודק האם אנו נדרשים להרחיב את המחסנית. המחסנית ריקה בתחילה, ואנו מבצעים n פעולות הכנסה. אם כל פעולת הרחבה מוסיפה k איברים, אז יהיה עלינו לבצע פעולות הרחבה. פעולת ההרחבה מתוארת בשורות 2-7, לכן שורות 2-7 ירוצו פעמים.

עבור פעולת הרחבה מספר i, שורות 2,5 יוצרות מחסניות חדשות ריקות בזמן ריצה לינארי (לפי i),

והלולאות בשורות 3,6 רצות length[T]=i\*k פעמים, כמספר איברי המערך, ובכל פעם מבצעות מספר פעולות קבוע. ז"א בסה"כ בתוכנית, שורות 2-7 ירוצו יחד (כאשר m זהו הקבוע שב-).

משום שאנו מנתחים סיבוכיות במקרה הגרוע, נוכל להניח ש מספר שלם, שכן ערך התקרה איננו משפיע על הסיבוכיות האסימפטוטית. ולכן - .

מתקיים:

אם k קבוע אז הנ"ל שווה ל- . אם k הוא משתנה בלתי תלוי ב-n, אז הנ"ל שווה ל.

בכל ריצה, שורה 8 רצה בסיבוכיות קבועה. לפיכך, בn פעולות הכנסה, היא תבוצע בסיבוכיות לינארית.

לסיכום, מתקיים:

* עבור k קבוע, .
* עבור k משתנה בלתי-תלוי ב-n מתקיים . ז"א:
  + אם אז .
  + אם , אז .

1. נניח שרצף פעולות ההכנסה מתחיל בכך שהמחסנית S מאותחלת רק עם האיבר הראשון.

לא ניתן לבצע את דרישת השאלה המדויקת (להכיל ממחסנית ריקה, ולבצע פעולת הרחבה של פי שניים יותר איברים, משום שאז ממערך עם 0 איברים נגדל למערך עם 0 איברים ולא נוכל לבצע את המשימה)

(ההבדלים משגרת סעיף א' מסומנים במרקר צהוב. תיעודי קוד זהים נשמטו בפתרון זה.)

Q3-STACK-PUSH-EXTEND-DOUBLE(S, x)

1 if top[S] == length[S]

2 then T 🡨 CREATE-STACK(length[S])

3 while not STACK-EMPTY(S)

4 do PUSH(T, POP(S))

► Create a new stack of size 2\*N

5 S 🡨 CREATE-STACK(2\*length[S])

6 while not STACK-EMPTY(T)

7 do PUSH(S, POP(T))

8 PUSH(S, x)

9 return S

ננתח את זמן הריצה עבור סדרה של n פעולות הכנסה החל ממחסנית ריקה:

כמצוין בעמוד 169 בספר הלימוד, כל אחת מהפעולות STACK-EMPTY, PUSH, POP מתבצעת בזמן קבוע.

תנאי 1 בודק האם אנו נדרשים להרחיב את המחסנית. המחסנית ריקה בתחילה, ואנו מבצעים n פעולות הכנסה. אם כל פעולת הרחבה מכפילה ב-2 את מספר האיברים, אז יהיה עלינו לבצע פעולות הרחבה. פעולת ההרחבה מתוארת בשורות 2-7, לכן שורות 2-7 ירוצו פעמים.

עבור פעולת הרחבה מספר i, שורות 2,5 יוצרות מחסניות חדשות ריקות בזמן ריצה לינארי (לפי i), והלולאות בשורות 3,6 רצות length[T]= פעמים, כמספר איברי המערך, ובכל פעם מבצעות מספר פעולות קבוע. ז"א בסה"כ בתוכנית, שורות 2-7 ירוצו יחד

(כאשר k זהו הקבוע שב-).

משום שאנו מנתחים סיבוכיות במקרה הגרוע, נוכל להניח ש מספר שלם, שכן ערך התקרה איננו משפיע על הסיבוכיות האסימפטומטית. ולכן - . *מתקיים:*

1. *מבחינה אסימפטומטית, שיטה ג' רצה בסיבוכיות לינארית, וגם שיטה ב' רצה בסיבוכיות לינארית כאשר k משתנה בלתי-תלוי ב-n ומתקיים (\*). שיטה ב' במקרים אחרים, ושיטה א', הינן פולינומיות. לכן, עבור ערכי n הולכים וגדלים, היעילות של השיטות הלינאריות יתעלו על השיטות הפולינומיות.*

*עם זאת, חשוב לציין שמבחינת סיבוכיות מקום, סעיף ג', וסעיף ב' בתנאים (\*) הם הבזבזניים ביותר.*

*עבור סעיף ג', לדוגמה, בסוף תהליך הכנסת n האיברים, נשאר עם מערך באורך תאי זיכרון (במקרה הגרוע פירוש הדבר הוא שאורך המערך הוא כמעט כפול מהאורך הנדרש), בניגוד לסעיף א', בו אורך המערך הוא כמספר האיברים (היעיל ביותר מבחינת הזיכרון), או סעיף ב' שלא בתנאי (\*), בו נשאר עם מערך באורך .*

*(בפועל סיבוכיות המיקום היא כמעט כפולה, שכן אנו מקצים זיכרון למערך המדובר בשורה 5 אך גם* יוצרים מערך עזר בשורה 2, אך ברור שאסימפטומטית אין לכך משמעות, וכמובן לא במקרה זה, בה הבאנו את המספרים למטרות השוואה בלבד).

**שאלה 4:**

ראשית נבין את הגדרת המערך.

המערך החדש S[1..n] מכיל באיבר ה-i את גודל הרצף הגדול ביותר משמאל ל-A[i] (כולל A[i] עצמו), שבו A[i] הוא מקסימלי (\*).

בפתרון, ההגדרה של "רצף" היא רצף המקיים את המתואר ב(\*), ולא רצף מבחינת הגדרתו המתמטית.

כמו כן, בפתרון נשתמש בשגרות הבסיסיות של מחסנית, כמוגדרות בספר הלימוד בעמוד 169.

בנוסף, נניח שקיימת שגרת עזר CREATE-ARRAY היוצרת מערך חדש בגודל N, ושקיימת שגרת עזר CREATE-STACK היוצרת מחסנית חדשה בגודל N.

Q4-BUILD-ARRAY-S(A)

1 S 🡨 CREATE-ARRAY(length[A])

2 for i🡨1 to length[A]

3 do longest\_seq 🡨 1

4 for j🡨i-1 downto 1

5 do if (A[j] <= A[i])

6 then longest\_seq 🡨 longest\_seq+1

7 else

8 then break

9 S[i] 🡨 longest\_seq

10 return S

נראה בקצרה את נכונות השגרה:

בשורה 1 אנו מקצים זיכרון למערך החדש S.

בלולאה שבשורות 2-9 אנו בונים את המערך החדש. בכל איטרציה i אנו בונים את התא i במערך S.

בשורות 3-9 אנו רצים לצד שמאל על המערך A, החל מהאיבר A[i] עצמו, ועד התחלת המערך, או עד שישבר הרצף.

בשורה 3, הרצף הארוך ביותר מוגדר להיות 1, שהרי A[i] עצמו ודאי קטן או שווה לעצמו. לאחר מכן, בלולאה שבשורות 4-9 אנו רצים עם האינדקס j, מi-1 ועד 1. האינדקס j מתקדם ע"י הפחתה ב-1.

בכל איטרציה בלולאה הפנימית, אנו בודקים האם A[j] <= A[i].

אם כן, פירוש הדבר הוא שהאיבר A[j] מצטרף לרצף, ועל כן אנו נגדיל את longest\_seq ונמשיך בריצה. אם לא, סימן שנשבר הרצף, ואנו יוצאים מן הלולאה.

לאחר שהלולאה הפנימית מסתיימת, longest\_seq מכיל את גודל הרצף(\*) הגדול ביותר.

בשורה 9 ערך זה נכנס אל מקומו המתאים במערך S.

ננתח את סיבוכיות השגרה:

יהי n גודל המערך A.

שורה 1 מקצה מערך חדש בזמן לינארי.

הלולאה החיצונית בשורות 2-9 רצה n פעמים.

בכל איטרציה i, הלולאה הפנימית בשורות 4-9 רצה במקרה הגרוע i-1 פעמים, ומבצעת בכל פעם מספר פעולות קבוע. לכן, הלולאה הפנימית רצה בסה"כ בסיבוכיות של:

לפיכך השגרה כולה רצה בסיבוכיות של:

*כנדרש.*

Q4-BUILD-ARRAY-S-VER2(A)

1 B 🡨 CREATE-ARRAY(length[A])

2 S 🡨 CREATE-STACK(length[A])

3 for i 🡨1 to n

4 do while (not STACK-EMPTY(S)) and (A[i] >= A[TOP(S)])

► We deliberately throw away the head value of S.

5 do POP(S)

6 if STACK-EMPTY(S)

7 then k 🡨0

8 else

9 then k 🡨 TOP(S)

10 S[i] 🡨 i-k

11 PUSH(D, i)

12 return S

נראה בקצרה את נכונות השגרה:

רעיון האלגוריתם הכללי הוא שאנו רצים על המערך A, ובכל פעם "דוחפים" את האינדקס הקודם אל מחסנית העזר.

בתחילת כל איטרציה אנו מנסים למצוא את ערך ה-k שעבורו i-k יקיים את התנאי (\*). במילים אחרות, אנו מחפשים את אורך הרצף המקסימלי (), שמסתיים באיבר A[i].

מחסנית העזר מכילה בתחילת כל איטרציה את ה"מכשולים" שA[i] צריך לעבור, ז"א, את אינדקסי האיברים שיש לבדוק האם A[i] גדול מהם. בלולאה בשורה 4 אנו "זורקים" את האינדקסים שקטנים או שווים לA[i], ז"א, אלה שנמצאים בתוך הרצף. הלולאה תפסיק כאשר המחסנית ריקה, או כאשר נתקלנו באיבר ששבר את הרצף.

בשורות 6-9 אנו קובעים את ערך ה-k הרלוונטי.

אם המחסנית ריקה, הרי שעברנו את כל ה"מכשולים" וA[i] הוא הגדול ביותר מתחילת המערך עד אליו. לכן k=0. במקרה זה, i-k=i, ואכן אורך הרצף הארוך ביותר הוא i. (שורה 7)

אם המחסנית איננה ריקה, אז ראש המחסנית הוא האינדקס הראשון ששבר את תנאי הרצף. לפיכך, אם אינדקס זה הוא x, אז k=x ואז i-k יהיה אורך הרצף הארוך ביותר. (שורה 9)

בשורה 10 אנו מכניסים את i-k, הלא הוא ערך הרצף המקסימלי, אל S[i].

בשורה 11 אנו מכניסים את האינדקס הנוכחי למחסנית, כדי שבאיטרציה הבאה, נשווה את האיבר הבא אל מול הנוכחי.

ננתח את סיבוכיות השגרה:

יהי n גודל המערך A.

כמצוין בעמוד 169 בספר הלימוד, כל אחת מהפעולות STACK-EMPTY, PUSH, POP מתבצעת בזמן קבוע.

שורות 1-2 מקצה מערך ומחסנית חדשים בזמן לינארי.

הלולאה החיצונית בשורות 3-14 רצה n פעמים.

נבחין ראשית שהמחסנית ריקה בתחילה, ומוכנסים אליה איברים רק בשורה 14. שורה 14 רצה פעם אחת בכל איטרציה של הלולאה החיצונית בשורות 3-14, ז"א בסה"כ במהלך כל השגרה, שורה 14 רצה n פעמים.

לפיכך אל המחסנית מוכנסת בשגרה כולה n איברים.

הלולאה הפנימית בשורות 4-8 רצה במקרה הגרוע בכל זמן ריצת השגרה n פעמים (שכן היא רצה במקרה הגרוע עד שהמחסנית מתרוקנת, שזהו מקסימום של n איברים). בכל איטרציה הלולאה מבצעת עבודה בזמן ריצה קבוע. לפיכך הלולאה הפנימית בשורות 4-8 רצה בזמן ריצה לינארי.

שורות 9-14 רצות בסיבוכיות קבועה, ומשום שהלולאה החיצונית בשורות 3-14 רצה n פעמים, שורות 9-14 רצות בכל השגרה בסיבוכיות לינארית.

לפיכך השגרה כולה רצה בסיבוכיות של:

*כנדרש.*