שאלה 1

א'

נניח בשלילה שקיים אלגוריתם מיון מבוסס השוואות המבצע לכל היותר 6 השוואות במקרה הגרוע עבור מערך באורך 5.

נסמן - גובה של עץ החלטה המתאים לאלגוריתם הנ"ל.

מההנחה נובע

עץ בינארי בגובה מכיל לכל היותר עלים, לכן בעץ ההחלטה יש לכל היותר עלים.  
אבל כל תמורה של מערך הקלט צריכה להופיע כעלה, ולמערך באורך 5 (שכל איבריו שונים) יש תמורות, כלומר בעץ ההחלטה חייבים להיות לפחות 120 עלים - סתירה למספר העלים שקיבלנו.

מסקנה: כל אלגוריתם מיון מבוסס השוואות הממיין מערך באורך 5 חייב לבצע לפחות 7 השוואות במקרה הגרוע.

ב'

SORT5(A):  
 if A[1] > A[2] then  
 exchange A[1]**↔**A[2]  
 if A[3] > A[4] then  
 exchange A[3]**↔**A[4]  
 if A[2] > A[4] then  
 exchange A[1]**↔**A[3]  
 exchange A[2]**↔**A[4]  
 a1🡨A[1]  
 a2🡨A[2]  
 a3🡨A[3]  
 a4🡨A[4]  
 a5🡨A[5]  
// Now a1 ≤ a2, a3 ≤ a4, a2 ≤ a4  
// We will sort all elements except for a3  
 if a5 ≤ a2 then  
 if a5 ≤ a1 then  
 SET-ARRAY5(A, a5, a1, a2, a4, 0)  
 else  
 SET-ARRAY5(A, a1, a5, a2, a4, 0)  
 else  
 if a5 ≤ a4 then  
 SET-ARRAY5(A, a1, a2, a5, a4, 0)  
 else  
 SET-ARRAY5(A, a1, a2, a4, a5, 0)  
// Now a1, a2, a3, a4 are correctly sorted internally, need to "push" a3 to its correct position  
 if a3 ≤ A[2] then  
 if a3 ≤ A[1] then  
 SET-ARRAY5(A, a3, A[1], A[2], A[3], A[4])  
 else  
 SET-ARRAY5(A, A[1], a3, A[2], A[3], A[4])  
 else  
 if a3 ≤ A[3] then  
 SET-ARRAY5(A, A[1], A[2], a3, A[3], A[4])  
 else  
 SET-ARRAY5(A, A[1], A[2], A[3], a3, A[4])

שגרת עזר

SET-ARRAY5(A, x1, x2, x3, x4, x5):  
A[1] = x1  
A[2] = x2  
A[3] = x3  
A[4] = x4  
A[5] = x5

ג'  
לא: עבור כל אלגוריתם מיון נציג קלט לדוגמה באורך 5 ומעקב אחרי האלגוריתם שמבצע מעל 7 השוואות.

**מיון-הכנסה** (עמ' 14 בספר) A=[5,4,3,2,1]:

j🡨2  
 key🡨4  
 i🡨1  
 \*compare A[1]>4\*  
 A[2]🡨5 // A=[5,5,3,2,1]  
 i🡨0  
 A[1]🡨4 // A=[4,5,3,2,1]  
j🡨3  
 key🡨3  
 i🡨2  
 \*compare A[2]>3\*  
 A[3]🡨5 // A=[4,5,5,2,1]  
 i🡨1  
 \*compare A[1]>3\*  
 A[2]🡨4 // A=[4,4,5,2,1]  
 i🡨0  
 A[1]🡨3 // A[3,4,5,2,1]  
j🡨4  
 key🡨2  
 i🡨3  
 \*compare A[3]>2\*  
 A[4]🡨5 // A[3,4,5,5,1]  
 i🡨2  
 \*compare A[2]>2\*  
 A[3]🡨4 // A[3,4,4,5,1]  
 i🡨1  
 \*compare A[1]>2\*  
 A[2]🡨3 // A[3,3,4,5,1]  
 i🡨0  
 A[1]🡨2 // A[2,3,4,5,1]  
j🡨5  
 key🡨1  
 i🡨4  
 \*compare A[4]>1\*  
 A[5]🡨5 // A[2,3,4,5,5]  
 i🡨3  
 \*compare A[3]>1\*  
// more than 7 compares, stopping algorithm.

**מיון-מיזוג** (עמ' 27 בספר) A=[1,2,4,3,5]:

MERGE-SORT(A,1,5):  
 1<5  
 q🡨3  
 MERGE-SORT(A,1,3):  
 1<3  
 q🡨2  
 MERGE-SORT(A,1,2):  
 1<2  
 q🡨1  
 MERGE-SORT(A,1,1):  
 MERGE-SORT(A,2,2):  
 MERGE(A,1,1,2):  
 n1🡨1  
 n2🡨1  
 create arrays L[1..2] R[1..2]  
 i🡨1  
 L[1]🡨1  
 j🡨1  
 R[1]🡨2  
 L[2]🡨∞  
 R[2]🡨∞  
 i🡨1  
 j🡨1  
 k🡨1  
 L[1]<=R[1] \*compare\*  
 A[1]🡨1  
 i🡨2  
 k🡨2  
 R[1]<L[2]  
 A[2]🡨2  
 j🡨2  
 // Now A=[1,2,4,3,5]  
 MERGE-SORT(A,3,3):  
 MERGE(A,1,2,3):  
 n1🡨2  
 n2🡨1  
 create array L[1..3] R[1..2]  
 i🡨1  
 L[1]🡨1  
 i🡨2  
 L[2]🡨2  
 j🡨1  
 R[1]🡨4  
 L[3]🡨∞  
 R[2]🡨∞  
 i🡨1  
 j🡨1  
 k🡨1  
 L[i]<=R[j] \*compare\*  
 A[1]🡨1  
 i🡨2  
 k🡨2  
 L[i]<=R[j] \*compare\*  
 A[2]🡨2  
 i🡨3  
 k🡨3  
 R[j]<L[i]  
 A[3]🡨4  
 j🡨2  
 // Now A=[1,2,4,3,5]  
 MERGE-SORT(A,4,5):  
 4<5  
 q🡨4  
 MERGE-SORT(A,4,4):  
 MERGE-SORT(A,5,5):  
 MERGE(A,4,4,5):  
 n1🡨1  
 n2🡨1  
 create array L[1..2] R[1..2]  
 i🡨1  
 L[1]🡨3  
 j🡨1  
 R[1]🡨5  
 L[2]🡨∞  
 R[2]🡨∞  
 i🡨1  
 j🡨1  
 k🡨4  
 L[1]<=R[1] \*compare\*  
 A[4]🡨3  
 i🡨2  
 k🡨5  
 R[1]<L[2]  
 A[5]🡨5  
 j🡨2  
 // Now A=[1,2,4,3,5]  
 MERGE(A,1,3,5):  
 n1🡨3  
 n2🡨2  
 create array L[1..4] R[1..3]  
 i🡨1  
 L[1]🡨1  
 i🡨2  
 L[2]🡨2  
 i🡨3  
 L[3]🡨4  
 j🡨1  
 R[1]🡨3  
 j🡨2  
 R[2]🡨5  
 L[4]🡨∞  
 R[3]🡨∞  
 i🡨1  
 j🡨1  
 k🡨1  
 L[1]<=R[1] \*compare\*  
 A[1]🡨1  
 i🡨2  
 k🡨2  
 L[2]<=R[1] \*compare\*  
 A[2]🡨2  
 i🡨3  
 k🡨3  
 R[1]<L[3] \*compare\*  
 A[3]🡨3  
 j🡨2  
 k🡨4  
 L[3]<=R[2] \*compare\*  
 A[4]🡨4  
 i🡨4  
 k🡨5  
 R[2]<L[4]  
 A[5]🡨5  
 j🡨3

יותר מ־7 השוואות.

**מיון-בחירה** (עמ' 13 במדריך) A=[1,2,3,4,5]:

SELECTION-SORT(A):  
 i🡨1  
 min🡨1  
 j🡨2  
 A[1]<A[2] \*compare\*  
 j🡨3  
 A[1]<A[3] \*compare\*  
 j🡨4  
 A[1]<A[4] \*compare\*  
 j🡨5  
 A[1]<A[5] \*compare\*  
 exchange A[1]**↔**A[1] // A=[1,2,3,4,5]  
 i🡨2  
 min🡨2  
 j🡨3  
 A[2]<A[3] \*compare\*  
 j🡨4  
 A[2]<A[4] \*compare\*  
 j🡨5  
 A[2]<A[5] \*compare\*  
 exchange A[2]**↔**A[2] // A=[1,2,3,4,5]  
 i🡨3  
 min🡨3  
 j🡨4  
 A[3]<A[4] \*compare\*  
// more than 7 compares, stopping algorithm.

**מיון-בועות** (עמ' 32 בספר) A=[1,2,3,4,5]:

BUBLESORT(A):  
 i🡨1  
 j🡨5  
 A[5]>A[4] \*compare\*  
 j🡨4  
 A[4]>A[3] \*compare\*  
 j🡨3  
 A[3]>A[2] \*compare\*  
 j🡨2  
 A[2]>A[1] \*compare\*  
 i🡨2  
 j🡨5  
 A[5]>A[4] \*compare\*  
 j🡨4  
 A[4]>A[3] \*compare\*  
 j🡨3  
 A[3]>A[2] \*compare\*  
 j🡨2  
 A[2]>A[1] \*compare\*  
// more than 7 compares, stopping algorithm.

**מיון-ערמה** (עמ' 113 בספר) A=[5,4,3,2,1]:

HEAPSORT(A):  
 BUILD-MAX-HEAP(A):  
 heap-size[A]🡨5  
 i🡨2  
 MAX-HEAPIFY(A, 2):  
 l🡨4  
 r🡨5  
 l<=heap-size[A], A[l]<A[i] \*compare\*  
 largest🡨2  
 r<=heap-size[A], A[r]<A[largest] \*compare\*  
 largest=i  
 i🡨1  
 MAX-HEAPIFY(A, 1):  
 l🡨2  
 r🡨3  
 l<=herap-size[A], A[l]<A[i] \*compare\*  
 largest🡨1  
 r<=heap-size[A], A[r]<A[largest] \*compare\*  
 largest=i  
 i🡨5  
 exchange A[1]**↔**A[5] // A=[1,4,3,2,5]  
 heap-size🡨4  
 MAX-HEAPIFY(A,1):  
 l🡨2  
 r🡨3  
 l<=heap-size[A], A[l]>A[i] \*compare\*  
 largest🡨2  
 r<=heap-size[A], A[r]<A[largest] \*compare\*  
 largest≠i  
 exchange A[1]**↔**A[2] // A=[4,1,3,2,5]  
 MAX-HEAPIFY(A,2):  
 l🡨4  
 r🡨5  
 l<=heap-size[A], A[l]>A[i] \*compare\*  
 largest🡨4  
 r>heap-size[A]  
 largest≠i  
 exchange A[2]**↔**A[4] // A=[4,2,3,1,5]  
 MAX-HEAPIFY(A,4):  
 l🡨8  
 r🡨9  
 l>heap-size[A]  
 largest=4  
 r>heap-size[A]  
 largest=i  
 i🡨4  
 exchange A[1]**↔**A[4] // A=[1,2,3,4,5]  
 heap-size[A]🡨3  
 MAX-HEAPIFY(A,1):  
 l🡨2  
 r🡨3  
 l<=heap-size, A[l]>A[i] \*compare\*  
// more than 7 compares, stopping algorithm.

**מיון-מהיר** (עמ' 122 בספר) A=[5,4,3,2,1]:

QUICKSORT(A,1,5):  
 1<5  
 PARTITION(A,1,5):  
 x🡨1  
 i🡨0  
 j🡨1  
 A[1]>1 \*compare\*  
 j🡨2  
 A[2]>1 \*compare\*  
 j🡨3  
 A[3]>1 \*compare\*  
 j🡨4  
 A[4]>1 \*compare\*  
 exchange A[1]**↔**A[5] //A=[1,4,3,2,5]  
 return 1  
 q🡨1  
 QUICKSORT(A,1,0):  
 1>0  
 QUICKSORT(A,2,5):  
 2<5  
 PARTITION(A,2,5):  
 x🡨5  
 i🡨1  
 j🡨2  
 A[4]<=5 \*compare\*  
 i🡨2  
 exchange A[2]**↔**A[2]  
 j🡨3  
 A[3]<=5 \*compare\*  
 i🡨3  
 exchange A[3]**↔**A[3]  
 j🡨4  
 A[4]<=5 \*compare\*  
 i🡨4  
 exchange A[4]**↔**A[4]  
 exchange A[5]**↔**A[5]  
 return 5  
 q🡨5  
 QUICKSORT(A,2,4):  
 2<4  
 PARTITION(A,2,4):  
 x🡨2  
 i🡨1  
 j🡨2  
 A[4]>2 \*compare\*  
// more than 7 compares, stopping algorithm.

שאלה 2

א'

DETERMINE-2-Z(S, n, k, z):  
 if k then  
 MERGE-SORT(S,1,n)  
 else  
 // See שאלה ז-18 במדריך  
 Write each number in S as k digits in base n  
 RADIX-SORT(S,k) using COUNTING-SORT  
 // Now S is sorted  
 i🡨1  
 j🡨n  
 while i<j  
 if S(i)+S(j) < z then  
 i🡨i+1  
 else if S(i)+S(j) > z then  
 j🡨j-1  
 else  
 return "yes"  
 return "no"

הסבר: קודם מבוצע מיון (אלגוריתם המיון נקבע לפי היחס בין k ל־n ע"פ שיקולי יעילות).  
לאחר שהמערך ממויין מאתחלים משתנים שיצביעו לתחילת וסוף המערך, וכל עוד הם לא נפגשו משווים את סכום האיברים שהם מצביעים עליהם ל־z. אם הוא שווה מסיימים עם תשובה חיובית, אם לא אז מגדילים את המצביע הקטן או מקטינים את הגדול בהתאם לצורך.  
ניתוח זמן ריצה: אם מבוצע MERGE-SORT שלוקח במקרה הגרוע, אם מבוצע האלגוריתם המתואר בשאלה ז-18 במדריך הלמידה שזמן ריצתו .  
שאר האלגוריתם מבצע מעבר יחיד על המערך - בזמן במקרה הגרוע.  
לכן זמן הריצה הכולל הוא

ב'

DETERMINE-3-Z(S,n,k,z):  
 MERGE-SORT(S,1,n)  
 for i🡨1 to n  
 j🡨1  
 k🡨n  
 while j<k  
 if S(i)+S(j)+S(k)<z then  
 j🡨j+1  
 else if S(i)+S(j)+S(k)>z then  
 k🡨k-1  
 else if i≠j AND i≠k then  
 return "yes"  
 return "no"

הסבר: מתבצע מיון, ולאחר מכן לכל אינדקס במערך (ערך של i) נבדק בלולאה פנימית אם יש שני איברים שסכומם בצירוף הערך באינדקס i שווה ל־z, בדומה לחלק א'.  
ניתוח זמן ריצה: MERGE-SORT -   
לולאה חיצונית - n פעמים במקרה הגרוע, לולאה פנימית - n פעמים במקרה הגרוע, לכן זמן הריצה הוא (המקרה הגרוע ביותר מתקיים כאשר לא קיימים שלושה איברים שסכומם z)

שאלה 3

נגדיר רשימה XL באופן הבא:  
head[XL] - מצביע לאיבר הראשון (או 0 אם הרשימה ריקה)  
tail[XL] - מצביע לאיבר האחרון (או 0 אם הרשימה ריקה)  
לכל איבר x ברשימה יהיו השדות הבאים:  
key[x] - ערך האיבר.  
np[x] - XOR בין המצביע לאיבר הקודם למצביע לאיבר הבא.

אם יש לנו מצביעים לשני איברים עוקבים x, y ניתן למצוא את האיבר הבא z כך:  
z=np[y] XOR x כי np[y]=x XOR z

את הפעולות המבוקשות ניתן לממש באופן הבא:

// In these algorithms, it is assumed that NIL=0  
SEARCH(XL, k):  
 prev🡨NIL  
 next🡨head[XL]  
 while next≠NIL AND key[next]≠k  
 newNext🡨prev XOR np[next]  
 prev🡨next  
 next🡨newNext  
 return next

הסבר: מעבר סדרתי על הרשימה בחיפוש אחר האיבר. מעבר מאיבר לאיבר ע"י XOR בהתאם להסבר למעלה.  
זמן ריצה:

INSERT(XL, x):  
 oldHead🡨head[XL]  
 if oldHead=NIL  
 tail[XL]🡨x  
 else  
 np[oldHead]🡨x XOR np[oldHead]  
 np[x]🡨oldHead  
 head[XL]🡨x

הסבר: הכנסת איבר לראש הרשימה. עדכון ערך np של האיבר החדש והאיבר הקודם בראש הרשימה. בנוסף אם הרשימה היתה ריקה, עדכון tail להצביע על האיבר החדש.  
זמן ריצה:

DELETE(XL, x):  
 prevprev=NIL  
 prev=NIL  
 curr=NIL  
 next=NIL  
 nextnext=head[XL]  
 while (curr≠x)  
 if nextnext≠NIL then  
 newNextnext🡨 next XOR np[nextnext]  
 else  
 newNextnext🡨NIL  
 prevprev🡨prev  
 prev🡨curr  
 curr🡨next  
 next🡨nextnext  
 nextnext🡨newNextnext  
 if prev=NIL then  
 head[XL]🡨next  
 else  
 np[prev]🡨prevprev XOR next  
 if next=NIL then  
 tail[XL]🡨prev  
 else  
 np[next]🡨prev XOR nextnext

הסבר: 5 מצביעים לאיברים עוקבים מעודכנים בלולאה, מתקדמים איבר איבר עד ש־curr שווה ל־x.  
מסירים את x מהרשימה ע"י עדכון שדה np של prev ו־next, וטיפול מתאים ב־head/tail במקרים שהאיבר ראשון/אחרון ברשימה.  
זמן ריצה:

// Flip the order of the elements  
FLIP(XL):  
 temp🡨head[XL]  
 head[XL]🡨tail[XL]  
 tail[XL]🡨temp

הסבר: החלפה בין ערכי head ו־tail של הרשימה. זה גורם להפיכת סדר האיברים.  
זמן ריצה:

שאלה 4

א'

BUILD-S-1(P):  
 n🡨length[P]  
 create array S[1..n]  
 for i🡨1 to n  
 j🡨i  
 while j>=1 AND P[j]<=P[i]  
 j🡨j-1  
 P[i]🡨i-j

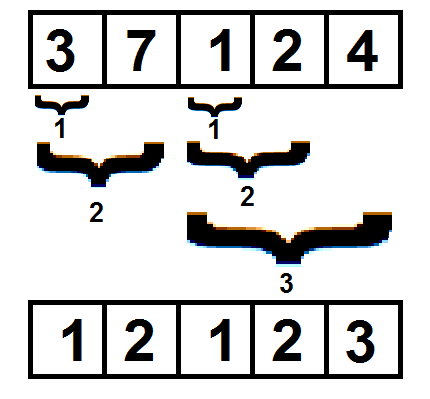
הסבר: לכל אינדקס 1..n האלגוריתם סורק את המערך P שמאלה עד שהוא מוצא איבר יותר גדול מ־P[i] או את תחילת המערך, ושם ב־B[i] את ההפרש בין האינדקסים (שזה כמות האיברים שצמודים ל־i משמאל ולא גדולים ממנו)

ניתוח זמן ריצה: הלולאה החיצונית תמיד תרוץ n פעמים.  
במקרה הגרוע (המערך בסדר לא־יורד), הלולאה הפנימית תרוץ i פעמים. הלולאה הפנימית תתבצע פעמים. לכן זמן הריצה במקרה הגרוע הוא .

ב'

BUILD-S-2(P):  
 n🡨length[P]  
 create array S[1..n]  
 indexes🡨Init stack  
 for j🡨n to 1  
 // STACK-TOP is defined in שאלה ח-2 במדריך  
 while (NOT STACK-EMPTY(indexes)) AND (P[j] > P[STACK-TOP(indexes)])  
 tmp🡨POP(indexes)  
 S[tmp]🡨tmp-j  
 PUSH(indexes, j)  
 while NOT STACK-EMPTY(indexes)  
 tmp🡨POP(indexes)  
 S[tmp]🡨tmp

הסבר כללי:   
סורקים את המערך מימין לשמאל, מכניסים כל אינדקס למחסנית. אם מגיעים לאיבר שיותר גדול מהאיבר שבאינדקס שבראש המחסנית (או לתחילת המערך), מוציאים אינדקס מראש המחסנית, ושמים במערך הפלט את ההפרש בין האינדקס מהמחסנית לאינדקס הנוכחי.

הדגמה

ניתוח זמן ריצה:  
הלולאה הראשית רצה n פעמים, ומבצעת כל פעם הכנסה אחת למחסנית. לולאות ה־while רצות רק כל עוד יש איברים במחסנית, ומבצעות שליפה אחת בכל איטרציה (סה"כ n שליפות, כמספר ההכנסות). שליפה והכנסה למחסנית לוקחות . לכן זמן הריצה של האלגוריתם כולו במקרה הגרוע הוא .