עבודה 2 במבני נתונים ואלגוריתמים
מבני נתונים בסיסיים ואבסטרקטיים, אלגוריתמי מיון מתקדמים

בעבודה זו עליכם לממש ויראציות של אלגוריתמי מיון מתקדמים שנלמדו בהרצאות ובתרגולים תוך כדי שימוש באוסף פעולות של מבני נתונים אבסטרקטיים. חלק מהפעולות של מבני נתונים אלו יהיה עליכם לממש בעצמכם, ובחלק תוכלו להשתמש ללא מימוש (מימוש כבר קיים).

העבודה תמומש בשפת C. לעבודה זו מצורפים קבצי h. (הצהרה על אוסף פעולות) וקבצי c. (מימוש אוסף פעולות). ברוב קבצי העבודה אינכם צריכים להכניס שינויים, ולמעשה, אינכם רשאים לעשות זאת.

להלן פירוט ותאור רשימת הקבצים המתקבלת עם העבודה. אסור להוסיף קבצים נוספים.

LinkedList.h

תאור אוסף הפעולות של רשימה מקושרת שניתן להשתמש בהן בעבודה זו. **אין לבצע שינויים בקובץ זה.**

LinkedList.c

מימוש כל אוסף הפעולות המתוארות בקובץ LinkedList.h. **אין לבצע שינויים בקובץ זה. אין צורך לקרוא את תוכן הקובץ.**

Stack.h

תאור אוסף הפעולות של מחסנית שניתן להשתמש בהן בעבודה זו. **אין לבצע שינויים בקובץ זה.**

Stack.c

מימוש כל אוסף הפעולות המתוארות בקובץ Stack.h. **אין לבצע שינויים בקובץ זה.**

Queue.h

תאור אוסף הפעולות של תור שניתן להשתמש בהן בעבודה זו. **אין לבצע שינויים בקובץ זה.**

Queue.c

מימוש כל אוסף הפעולות המתוארות בקובץ Queue.h. חלק מהפעולות כבר ממומשות, חלק ממומשות חלקית, וחלק לא. עליכם להשלים את מימוש הפעולות החסרות בזמן הריצה המצויין בתיאור הפעולה כפי שמופיע בקובץ Queue.h.

MinPriorityQueue.h

תאור אוסף הפעולות של תור קדימויות מינימאלי שניתן להשתמש בהן בעבודה זו. **אין לבצע שינויים בקובץ זה.**

MinPriorityQueue.c

מימוש כל הפעולות המופיעות בקובץ MinPriorityQueue.h. חלק מהפעולות כבר ממומשות, חלק ממומשות חלקית, וחלק לא. על מנת לממש את הפעולות, קובץ זה מכיל גם פונקציות עזר – חלקן ממומשות ואת חלקן עליכם לממש. עליכם להשלים את מימוש הפעולות והפונקציות עזר החסרות בזמן הריצה המתואר בקבצים הללו.

SortingAlgorithms.h

קובץ זה מתאר את רשימת אלגוריתמי המיון שעליכם לממש בעבודה זו. **אין לבצע שינויים בקובץ זה.**

SortingAlgorithms.c

בקובץ זה עליכם לממש את כל אלגוריתמי המיון המתוארים בקובץ SortingAlgorithms.h. לשם מימוש חלק מהאלגוריתמים יהיה עליכם לממש פונקציות עזר נוספות המופיעות בקובץ זה, בזמן ריצה המתואר בקבצים הללו.

SortingAlgorithmsUtil.h
קובץ זה מכיל תאור פונקציות עזר כלליות שניתן לעשות בהן שימוש בעת מימוש הפעולות בקובץ SortingAlgorithms.c. **אין לבצע שינויים בקובץ זה.**

SortingAlgorithmsUtil.c
מימוש כל הפעולות המופיעות בקובץ SortingAlgorithmsUtil.h. **אין לבצע שינויים בקובץ זה. אין צורך לקרוא את תוכן הקובץ.**

Util.h

קובץ זה מכיל תאור פונקציות עזר כלליות שנעשה בהן שימוש בכל התוכנית. בפועל, פונקציית העזר היחידה בקובץ זה שתוכלו להיעזר בה היא SwapIndices. **אין לבצע שינויים בקובץ זה, פרט לשורה #define DEBUG – פרטים בהמשך.**

Util.c

מימוש כל הפעולות המופיעות בקובץ Util.h. **אין לבצע שינויים בקובץ זה. אין צורך לקרוא את תוכן הקובץ.**

assignment2.c

קובץ המכיל את פונקציית main ומנהל את ריצת התוכנית. **אין לבצע שינויים בקובץ זה. אין צורך לקרוא את תוכן הקובץ.**

סיכום ביניים:

- את הקבצים הבאים עליכם **לקרוא ולבצע בהם שינויים**:
  Queue.c, MinPriotityQueue.c, SortingAlgorithms.c
- את הקבצים הבאים עליכם **לקרוא בלבד:**
  LinkedList.h, Stack.h, Stack.c, Queue.h, MinPriorityQueue.h, SortingAlgorithms.h, SortingAlgorithmsUtil.h
- את הקבצים הבאים **אין צורך לקרוא כלל:**
  LinkedList.c, SortingAlgorithmsUtil.c, Util.c, assignment2.c
- **יוצא דופן:**

Util.h – בקובץ זה **קראו** אך ורק את התאור המופיע מעל השורה #define DEBUG, ואת תאור הפעולה SwapIndices. בקובץ זה ניתן **לשנות** רק את השורה #define DEBUG מ-0 ל-1 או להיפך. כאשר הערך הינו 1, תוכלו לראות הדפסות עזר למסך. כאשר הערך הוא 0, לא תראו הדפסות עזר.

תיאור העבודה:

בקבצים הרלוונטיים עליכם לממש את הפעולות ופונקציות העזר החסרות. את מימוש הפעולות ופונקציות העזר יש לבצע בין השורות הבאות, **ואך ורק בין השורות הללו:**

```
/* YOUR CODE STARTS HERE */

/* YOUR CODE ENDS HERE */
```

בחלק מהפונקציות, חלק מהמימוש כבר קיים. בפרט, הצהרות על כל המשתנים שתצתרכו להשתמש בהם כבר קיימות בקובץ, **וחל איסור לשנות את ההצהרות הללו אלא אם נתקבל אישור מיוחד.**

Queue.c

קובץ זה מכיל את מימוש הפעולות של מבנה הנתונים "תור". תאור הפעולות מופיע בקובץ ה-h. המקביל. התור מומש באמצעות רשימה מקושרת (dataList):

```
struct queue {
    List * dataList;
    int limit;
    int used;
};
```

בנוסף, התור מוגבל להכלת limit איברים בלבד. המשתנה used מתאר כמה איברים נמצאים כרגע בתור. לרוב, בתור הממומש ע"י רשימה מקושרת אין מגבלת מקום. אולם, בתור זה יש מגבלת מקום על מנת להקשות עליכם במימוש חלק מהאלגוריתמי המיון.
עליכם למלא את המימושים החסרים בקובץ זה תוך שימוש בפעולות הניתנות לביצוע על רשימה מקושרת (ראו LinkedList.h).
זמן הריצה של הפעולות שהינכם ממששים צריך להיות כפי שרשום בתאור הפעולה. תאור הפעולה מופיע בקובץ ה-h.

MinPriorityQueue.c

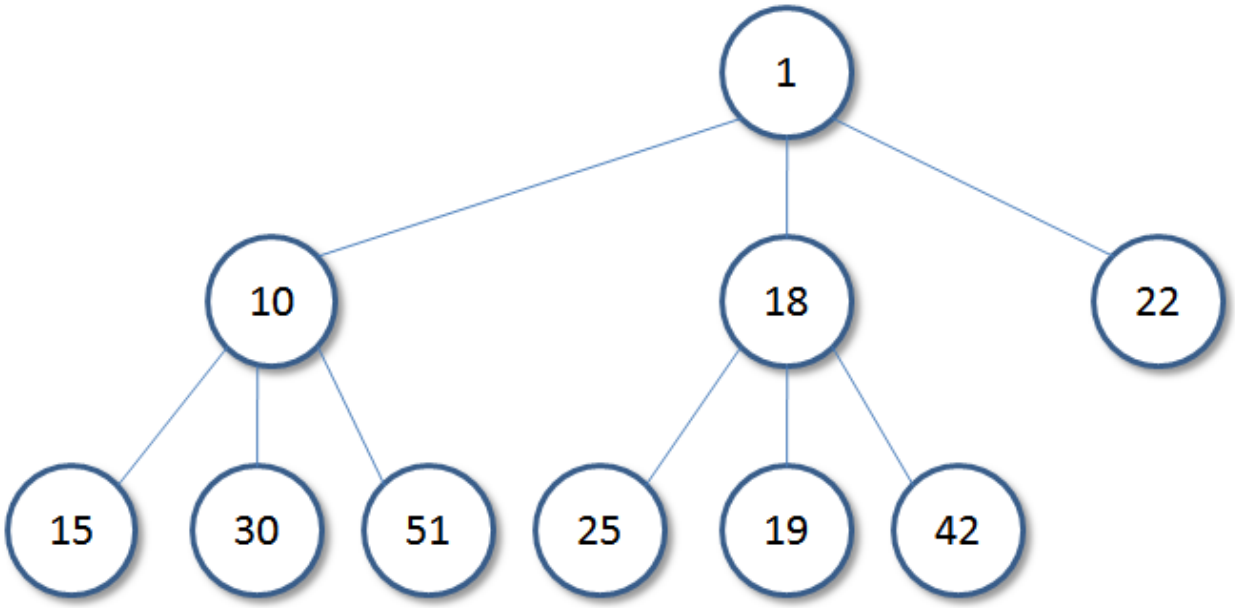
קובץ זה מכיל את מימוש הפעולות של מבנה הנתונים "תור קדימויות מינימאלי". תור קדימויות מינימאלי מומש באמצעות ערימת מינימום, שהיא ההפך מערימת מקסימום שראיתם בהרצאות ובתרגולים – כלומר, ערימה בה האבא של כל איבר **קטן** או שווה לו.

```
struct minPriorityQueue {
    int * arr;
    int arity;
    int limit;
    int used;
};
```

arr הוא המערך באמצעותו ממומשת הערימה. limit הוא הגודל המקסימאלי של המערך, ו-used מתאר את מספר האיברים שנמצאים בערימה.
ערימה זו היא ערימה מיוחדת – במקום שמספר הבנים של כל קודקוד בעץ המיוצג על ידי

הערימה יהיה 2, כפי שראיתם בהרצאות ובתרגולים, מספר הבנים של כל קודקוד בעץ ניתן בעת יצירת התור קדימויות ונשמר על ידי המשתנה arity. משתנה זה משפיע על מימוש חלק מהפעולות ופונקציות העזר שעליכם לממש בקובץ זה.
בעת מימוש הפעולות, ייתכן ותצתרכו, פרט לעדכון המערך arr, לעדכן גם את ערך המשתנה used.

דוגמא לערימת מינימום בעלת ערך arity=3 ובה used=10:



זמן הריצה של הפעולות שהינכם ממששים צריך להיות כפי שרשום בתאור הפעולה או פונקציית העזר המקומית. תאור הפעולה מופיע בקובץ ה-h. תאור פונקציות העזר מופיע מעל למימוש הפונקציה.

SortingAlgorithms.c

בקובץ זה עליכם לממש מספר אלגוריתמי מיון. תיאור האלגוריתמים מופיע גם ב-SortingAlgorithms.h.

וריאציות של מיון ערימה:

```
void AscendingHeapSort(int* arr, int size, int arity);
void DescendingHeapSort(int* arr, int size, int arity);
```

מימוש תור הקדימויות מאפשר לנו לממש את מיון הערימה בקלות באמצעות הפעולות של תור הקדימויות. עליכם לממש שתי וריאציות – אחת בה המערך ממויין בסדר עולה, ואחת בה המערך ממויין בסדר יורד. שימו לב שיצירת תור הקדימויות בו תשתמשו כבר נעשתה עבורכם. **אין לשנות את יצירת תור הקדימויות או ליצור תור קדימויות נוסף. חובה להשתמש בתור הקדימויות במימוש האלגוריתמים הללו.**
על האלגוריתמים לפעול בזמן המצויין בקובץ התיאור (.h). קחו בחשבון את זמן הריצה של פעולות תור הקדימויות כאשר הינכם מחשבים את זמן הריצה שלכם.

```
void PrintMinPQ(MinPriorityQueue* minpq);
```

בעת מימוש האלגוריתם, מומלץ להשתמש בפונקציה הנ"ל כפי שמתוארת ב-MinPriorityQueue.h לשם מציאת בעיות ותיקון תקלות בקוד שלכם במידה והוא איננו עובד כפי שציפיתם. פונקציה זו תעבוד רק כאשר ערך DEBUG ב-Util.h איננו 0.

וריאציות של מיון מהיר:

וריאציות איטרטיביות:

```
void StackBasedQuickSort(int* arr, int size);
void QueueBasedQuickSort(int* arr, int size);
```

מיון מהיר כפי שראיתם בכיתה הינו אלגוריתם רקורסיבי בו מוצאים את מיקום איבר הציר הנוכחי ואז מבצעים שתי קריאות רקורסיביות לשני תתי המערכים שנותר למיין.
בשתי הוריאציות הללו עליכם לממש את מיון מהיר **ללא** ביצוע של קריאות רקורסיביות, כלומר, באופן **איטרטיבי**. לשם כך תעזרו במבנה נתונים. בוריאציה הראשונה תשתמשו במחסנית עזר, ובשנייה בתור עזר. המבנים נוצרו עבורכם עם מגבלת מקום – עליכם לממש את האלגוריתמים ללא חריגה ממגבלת המקום של המחסנית או התור. מגבלת המקום היא

$2 * \lceil log(size) \rceil$.

חלק מהמימוש של מיון מהיר כפי שראיתם בהרצאות ובתרגולים כולל את פונקציית Partition אשר מוצאת את מיקום איבר הציר הנוכחי, מעבירה את איבר הציר הנוכחי למיקום זה ומחזירה את המיקום. עליכם לממש את פונקציה זו בדיוק כפי שראיתם בהרצאות ובתרגולים ולהשתמש בה במימוש האלגוריתמים הללו.

אין לשנות את יצירת מבנה הנתונים (מחסנית\תור) או ליצור מבנה נתונים נוסף. חובה להשתמש במבנה הנתונים במימוש האלגוריתמים הללו.

על האלגוריתמים לפעול בזמן הריצה המצויין בקובץ התיאור (.h).

```
BOOL IsValidSubArray(int p, int r, int size);
void MaybePushIndices(Stack* stack, int p, int r, int size);
void MaybeEnqueueIndices(Queue* queue, int p, int r, int size);
```

מומלץ להיעזר בפונקציות העזר הנ"ל. תאורן מופיע בקובץ SortingAlgorithmsUtil.h.

```
void PrintStackQuickSortState(Stack* stack, int* arr, int p, int q, int r, int size);
void PrintQueueQuickSortState(Queue* queue, int* arr, int p, int q, int r, int size);
void PrintPartitionState(int* arr, int size, int p, int r, int pivot, int q, int j);
```

בעת מימוש האלגוריתמים הללו, מומלץ להשתמש בפונקציות ההדפסה הנ"ל כפי שהן מתוארות ב-SortingAlgorithmsUtil.h לשם מציאת בעיות ותיקון תקלות בקוד שלכם במידה והוא איננו עובד כפי שציפיתם. פונקציה זו תעבוד רק כאשר ערך DEBUG ב-Util.h איננו 0.

<u>וריאציה המטפלת בערכים זהים (חוזרים):</u>

```
void EqualElementQuickSort(int* arr, int size, int p, int r);
```

בוריאציה זו של מיון מהיר ניתן להשתמש בגרסה הרקורסיבית כפי שראיתם בהרצאות ובתרגולים, אולם, את פעולת ה-Partition יש להחליף בפעולה חכמה יותר המפרידה את המערך לשלושה חלקים – איברים קטנים מאיבר הציר, איברים השווים לאיבר הציר, ואיברים הגדולים מאיבר הציר. על פעולה זו להחזיר את שתי קצוות תת-המערך בו נמצאים האיברים השווים לאיבר הציר, כך שהמיון המהיר יבצע קריאות רקורסיביות רק לתתי המערכים הכוללים איברים קטנים **ממש** מאיבר הציר או איברים גדולים **ממש** מאיבר הציר.

הדבר יכול לשפר משמעותית את זמן הריצה של מיון מהיר במקרים בהם ישנם הרבה ערכים שחוזרים על עצמם. קראו עוד כאן (עד Optimizations, לא כולל):
https://en.wikipedia.org/wiki/Quicksort#Repeated_elements

אין להשתמש במבנה נתונים עזר בשאלה זו.
על האלגוריתמים לפעול בזמן המצויין בקובץ התיאור (.h).

```
BOOL IsValidSubArray(int p, int r, int size);
```

מומלץ להיעזר בפונקציית העזר הנ"ל. תאורה מופיע בקובץ SortingAlgorithmsUtil.h.

```
void PrintSmartPartitionState(int* arr, int size, int p, int r,
                              int pivot, int q, int t, int j);
```

בעת מימוש האלגוריתם, מומלץ להשתמש בפונקציית ההדפסה הנ"ל כפי שהיא מתוארת ב-SortingAlgorithmsUtil.h לשם מציאת בעיות ותיקון תקלות בקוד שלכם במידה והוא איננו עובד כפי שציפיתם. פונקציה זו תעבוד רק כאשר ערך DEBUG ב-Util.h איננו 0.

<u>מיון בסיס:</u>

```
void nModuluRadixSort(int* arr, int size, int limit);
```

עליכם לממש מיון בסיס רגיל, אך הבסיס לחלוקת המיון איננו בהכרח דצימלי או בינארי, אלא מתקבל לפי הערך limit.
לדוגמא: כאשר אנו ממיינים לפי בסיס דצימלי, אנו ממיינים את ספרת האחדות, אח"כ את ספרת העשרות וכן הלאה. כאשר אנו ממיינים לפי בסיס בינארי, אנו ממיינים לפי הביט הכי פחות משמעותי של המספר (LSB – Least Significant Bit), אח"כ לפי הביט השני הכי פחות משמעותי, וכן הלאה. בוריאציה זו הבסיס איננו ידוע מראש. הינכם מקבלים את הבסיס באמצעות המשתנה limit.
אין להשתמש במבנה נתונים עזר בשאלה זו.
על האלגוריתמים לפעול בזמן המצויין בקובץ התיאור (.h).

```
void CountingSort(IntPair* arr, int size, int limit)
```

השתמשו במיון מנייה הנ"ל, שניתן לכם כשהוא כבר ממומש, על מנת למיין את האיברים בכל איטרציה בזמן לינארי ב-limit. מיון זה מקבל מערך מסוג IntPair – טיפוס זה מכיל שני

מספרים. המספר הראשון הוא המספר המקורי, והמספר השני הוא הערך לפיו יש למיין את המספר המקורי באיטרציה זו של מיון הבסיס.

```
void PrintnModuluRadixSortState(int* arr, IntPair* a, int size,
                                int limit, int denominator);
```

בעת מימוש האלגוריתם, מומלץ להשתמש בפונקציית ההדפסה הנ"ל כפי שהיא מתוארת ב-SortingAlgorithmsUtil.h לשם מציאת בעיות ותיקון תקלות בקוד שלכם במידה והוא איננו עובד כפי שציפיתם. פונקציה זו תעבוד רק כאשר ערך DEBUG ב-Util.h איננו 0.

התחלת הפרוייקט:

במודל תמצאו קובץ המדריך אתכם כיצד לפתוח פרוייקט חדש בויז'ואל סטודיו.
לאחר יצירת פרוייקט חדש, עליכם לשים בתיקיית הפרוייקט את כל קבצי הפרוייקט כפי שנמצאים במודל. את קבצי ה-h. יש לגרור לתוך תיקיית ה-Header של הפרוייקט בויז'ואל סטודיו, ואת קבצי ה-c. יש לגרור לתוך תיקיית ה-Source של הפרוייקט בויז'ואל סטודיו.
במודל תמצאו תמונה המסבירה את סדר הפעולות לאחר יצירת הפרוייקט הריק.
כעת תוכלו להתחיל את העבודה.

הרצת הפרוייקט:

לאחר ביצוע קומפילציה, תוכלו להריץ את הפרוייקט. תתבקשו לבחור אחת מבין שתי אופציות – הרצה ידנית של אלגוריתמים, או הרצת מבחן.

הרצה ידנית של אלגוריתמים:

באופציה זו עליכם לבחור את גודל המערך, טווח הערכים של המערך (מ-0 עד מספר לבחירתכם) ודרך יצירת המערך. כרגע ישנן 3 דרכים ליצירת המערך – ידנית, רנדומאלית ורנדומאלית בעלת ערך חוזר. מטרת הדרך השלישית היא לבחון את ההבדל בין ריצת אלגוריתמי המיון המהיר המשתמשים ב-Partition רגיל לעומת ריצת אלגוריתם המיון המהיר המשתמש ב-SmartPartition. בדרך זו במערך הנוצר רנדומאלית מובטח ערך אחד לפחות שיחזור על עצמו כמות משמעותית של פעמים.

ייתכן ובעתיד קובץ ה-assignment2.c יעודכן על מנת שהעבודה תכיל עוד דרכי יצירת מערכים, או שהדרכים הנוכחיות ישוכללו. כאשר קובץ זה יעודכן תקבלו עדכון מתאים. עדכון קובץ זה לא ישפיע על ריצת האלגוריתמים שלכם.

<u>הרצת מבחן</u>:

אופציה זו טרם ממומשת. עדכון עתידי לקובץ assignment2.c יאפשר לכם להריץ את אופציה זו. באופציה זו האלגוריתמים שלכם יורצו מספר גדול של פעמים, על מערכים בגדלים שונים וסוגים שונים. בתום ריצת המבחן יודפסו למסך זמני ריצה ממוצעים של האלגוריתמים שלכם על גדלים (וסוגים) שונים של מערכים.
לעבודה מצורף קובץ **exe** שהוא תוכנית בה כל האלגוריתמים ממומשים. תוכלו לבצע הרצת מבחן בקובץ זה ובתוכנית שלכם על מנת להשוות את זמני הריצה של האלגוריתמים שלכם לזה שבקובץ **exe** וכן כדי לוודא שהאלגוריתמים שלכם תמיד ממיינים נכונה את המערכים.

<u>הגשה:</u>

יש להגיש את שלושת הקבצים הבאים בלבד:

Queue.c, MinPriorityQueue.c, SortingAlgorithms.c

על הקבצים להיות מוגשים בתוך קובץ rar. או zip. ששמו הוא מספרי תעודות הזהות של המגישים המופרדים באמצעות קו תחתון. לדוגמא – 1111111_2222222.zip.

את העבודה יש להגיש **ביחידים או בזוגות בלבד.**

תאריך אחרון להגשה – 13.6.2017.

<u>סיכום נקודות חשובות:</u>

- עליכם להוסיף קוד אך ורק במקומות המיועדים לכך – בין היכן שכתוב שעל הקוד להתחיל והיכן שכתוב שעליו להסתיים.
- השורה היחידה בקוד שהיא יוצאת דופן מבחינה זו היא שורת ה-**#define DEBUG** הנמצאת בקובץ **Util.h**.

- שינויים נוספים בקוד ייתכנו באישור מיוחד בלבד.
- **אין להשתמש בפונקציות ספריה של שפת** C**.**
- אין צורך להצהיר על משתנים חדשים, כל המשתנים שתצתרכו להשתמש בהם הוצהרו כבר. אם אינכם מצליחים לממש באמצעות המשתנים הקיימים, ניתן להצהיר על משתנים חדשים, אך ורק בתוך האיזור המותר. בכל אופן **אסור** ליצור מבני נתונים נוספים, אלא רק משתנים פשוטים (כגון **int**).
- השתמשו בפונקציות עזר הנמצאות ב-**SortingAlgorithmsUtil.h**, או בתוך קבצי ה-**c.** עליהם אתם עובדים, במידה ויש בהם פונקציות עזר. בנוסף, ניתן להשתמש בפונקציה **SwapIndices** הנמצאת בקובץ **Util.h**.
- בפרט השתמשו בפונקציות ההדפסה, שיכולות לעזור לכם מאוד.
- הקפידו על זמני הריצה! על הפונקציות\פעולות\אלגוריתמים שלכם לפעול לפי זמני הריצה המצויינים.
- שאלות נא לפרסם בפורום העבודה במודל.