קורס יסודות התכנות בשפת C

פרק 16

חיפוש, מיון ומיזוג

Search, Sort & Merge

ד"ר שייקה בילו

יועץ ומרצה בכיר למדעי המחשב וטכנולוגית מידע

מומחה למערכות מידע חינוכיות, אקדמיות ומנהליות

מיון בועות למערך חד ממדי

2

הגדרת משתנה גלובלי לגודל המערך, הכרזות/הכרזות על שתי פונקציות - המיון וההדפסה.

```
#include <stdio.h>
#define SIZE 10
int sort(int numbers[ ]);
int print(int numbers[ ]);
int main()
{
  int numbers[SIZE];
  int i;
  printf("please enter 10 numbers:\n\n");
```

מיון בועות למערך חד ממדי

3

```
for(i = 0; i < SIZE; i++)
{
printf("Enter number in place %d \n",i+1);
scanf("%d",&numbers[i]);
}
        sort(numbers);
        print(numbers);
return 0;
}
```

תוכנית ראשית המפעילה את תהליך קליטת הערכים למערך

קריאה לפונקציית המיון

קריאה לפונקציית ההדפסה

מיון בועות למערך חד ממדי

4

```
int sort(int numbers[ ])
{
   int i, j, temp;
   for (i = (SIZE-1); i >= 0; i--)
        for (j = 1; j <= i; j++)
        {
             if (numbers[j-1] > numbers[j])
             {
                 temp = numbers[j-1];
                 numbers[j-1] = numbers[j];
                 numbers[j] = temp;
             }
        }
  return 0;
}
```

פונקציית מיון מבצעת את מיון הבועות למערך שנקלט בתוכנית הראשית

ביצוע מעבר על כל איברי המערך בלולאה כפולה כדי לוודא את היחס בין כל איבר לכל שכניו

מיון בועות למערך חד ממדי

פונקציית ההדפסה

```
int print(int numbers[], int s)
{
        int i;
        printf("\n\nSorted  array:\n\n");
        for(i = 0; i < SIZE; i++)
        {
                printf("Array place number %d  ",i);
                printf("Value %d  \n", numbers[i]);
        }
        return 0;
}
```

שאלות?

מיון בועות רקורסיבי למערך חד ממדי

הגדרת משתנה גלובלי לגודל המערך ביצוע הכרזות על שתי הפונקציות:

מיון והדפסה של המערכים

```
#include <stdio.h>
#define SIZE 10
int sortrec(int numbers[ ]);
int print(int numbers[ ]);
int main()
{
  int numbers[SIZE];
  int i;
  printf("please enter %d numbers:\n\n",SIZE);
```

מיון בועות רקורסיבי למערך חד ממדי


```
for(i = 0; i < SIZE; i++)
{
    printf("Enter number in place %d \n",i+1);
    scanf("%d",&numbers[i]);
}
  sortrec(numbers,SIZE);
  print(numbers);
  return 0;
}
```

תוכנית ראשית המפעילה את תהליך קליטת הערכים למערך

קריאה לפונקציית המיון

קריאה לפונקציית ההדפסה

מיון בועות רקורסיבי למערך חד ממדי


```
void swap(int *mis1, int *mis2)
{
        int temp;
        temp = *mis1;
        *mis1 = * mis2;
        *mis2 = temp;
}
```

פונקציית ההחלפה המקבלת כפרמטרים את שני המספרים ותוך שני מצביעים ומחליפה בין שני האיברים.

איברים אלה נשלחו לפונקציה כיוון שהם שכנים במערך המקורי והראשון גדול משכנו.

מיון בועות רקורסיבי למערך חד ממדי


```
int sortrec(int numbers[],int size)
{
   int i;
   if(size <= 1)
       return;
   for(i=0;i<size-1;i++)
        if(numbers[i]>numbers[i+1])
            swap(&numbers[i],&numbers[i+1]);
   sortrec(numbers,size-1);
return 0;
}
```

פונקציית המיון הרקורסיבית מבצעת את מיון הבועות למערך שנקלט בתוכנית הראשית תוך שימוש בפונקציית swap

ביצוע מעבר על כל איברי המערך בלולאה כדי לוודא את היחס בין כל איבר לכל שכניו וקריאה לפונקציית ההחלפה swap במידת הצורך

מיון בועות רקורסיבי למערך חד ממדי

פונקציית ההדפסה

```
void print(int numbers[])
{
        int i;
        printf("\n\nSorted  array:\n\n");
        for(i = 0; i < SIZE; i++)
        {
                printf("Array place number %d  ", i);
                printf("Value %d  \n", numbers[i]);
        }
}
```

שאלות?

חיפוש לינארי במערכים

נתון מערך ממוין בגודל 12 נרצה לחפש בו איבר כלשהו.

- חיפוש ליניארי

5	8	9	12	15	17	20	21	24	27	29	31

נחפש את המספר 17 חיפוש רגיל, לינארי, על כל איברי המערך הממוין, החל מהאיבר הראשון ועד לאיבר האחרון ונעצור ברגע שנמצא אותו

פתרון חיפוש לינארי במערכים


```
#include  <stdio.h>
#define SIZE 12
int linarySearch(int arr[], int item )
{
   int i;
   for(i=0;i<SIZE;i++)
   {
         if (arr[i] == item)
         {
           printf("The item %d in place %d",arr[i], i+1);
          return 1;
         }
   }
  return 0;
}
```

ביצוע מעבר על כל איברי המערך בלולאה כדי לוודא איפה, אם בכלל, נמצא ה- item אותו אנחנו מחפשים

במידה והאיבר item נמצא מדפיסים הודעה למסך, במידה ולא הפונקציה תחזיר 0

התוכנית הראשית


```
int main()
{
        int arr[SIZE]={5,8,9,12,15,17,20,21,24,27,29,31};
        int item=17,status=0;
        status=linarySearch( arr,item);
        if(status)
                printf("item is found in array!!!\n");
        else
                printf("item is not found in array!!!\n");
        return 0;
}
```

שאלות?

חיפוש בינארי במערכים

נתון מערך ממוין נרצה לחפש בו איבר כלשהו.

- **חיפוש בינארי**

5	8	9	12	15	17	20	21	24	27	29	31

נחפש את המספר 24 חיפוש בינארי, על כל איברי המערך הממוין, נתחיל בחציית המערך לחצי ובדיקה מה היחס בין האיבר אותו מחפשים לאיבר האחרון בחציון וכך הלאה עד למציאת האיבר

חיפוש בינארי במערכים

5	8	9	12	15	17	20	21	23	24	29	31

נחפש את 24 חיפוש בינארי כך שנחצה את המערך לשניים ונבדוק האם 24 גדול מהאיבר האחרון במחצית כך נדע האם להמשיך שמאלה או ימינה בחיפוש.

במידה ולא נמצא נמשיך לחצות את החצי בו הוא עשוי להיות ושוב נחפש אותו עד למציאתו במקום בו הוא קיים

פתרון חיפוש בינארי במערכים


```
#include  <stdio.h>
#define SIZE 12
int binarySearch(int arr[ ], int item )
{
      int lsize=0, rsize=SIZE-1, middle;
      while ( lsize <= rsize )
      {
         middle = (rsize + lsize )/2;
         if (item == arr[middle])  return 1;
         if (item < arr[middle])
             rsize = middle-1;
         else
             lsize = middle+1;
      }
return 0; }
```

התוכנית הראשית


```
int main()
{
        int arr[SIZE]={5,8,9,12,15,17,20,21,24,27,29,31};
        int item=17, status=0;
        status=binarySearch(arr,item);
        if(status == 1)
                printf("item is found in array!!!\n");
        else
                printf("item is not found in array!!!\n");
        return 0;
}
```

שאלות?

פתרון חיפוש בינארי רקורסיבי במערכים


```
#include  <stdio.h>
#define SIZE 12
int binarySearch(int arr[ ], int n, int item )
{
   int pos;
   if(arr[0] == item)
         return 0;
   if(n==0)
         return -1;
```

פתרון חיפוש בינארי רקורסיבי במערכים


```
if(arr[n/2] == item)
        return n/2;
if(arr[n/2] > item)
        return binarySearch(arr, n/2, item );
  else
   {
        pos = binarySearch(arr+n/2+1, n-n/2-1, item );
       if(pos == -1)
           return -1;
      return pos + n/2 +1;
   }
}
```

תוכנית ראשית - חיפוש בינארי רקורסיבי


```
int main()
{
    int arr[SIZE]={5,8,9,12,15,17,20,21,23,24,29,31};
    int item=24, status=0;
    status=binarySearch(arr,SIZE,item);
    if(status != -1)
        printf("\n%d is found in array in place %d!!!\n",item,status);
   else
        printf("\nItem %d is not found in array!!!\n“,item);
return 0;
}
```

שאלות?

תרגיל 4 - מיון בועות למערך דו ממדי


```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define ROWS 5
#define COLUMN 5
/*function prototype*/
void printArray(int [][COLUMN]);
void initRandomValues (int [][COLUMN]);
void bubblesort(int [][COLUMN]);
```

מיון בועות למערך דו ממדי


```
void main()
{
        int arr[ROWS][COLUMN];
        srand(time(NULL));
        initRandomValues(arr);
        printf("The array before bubble sort:\n");
        printf("=======================\n");
        printArray (arr);
        bubblesort(arr);
        printf("The array after bubble sort:\n");
        printf("=======================\n");
        printArray(arr);
}
```

מיון בועות למערך דו ממדי


```
void initRandomValues(int arr[][COLUMN])
{
    int i,j;
    for(i=0;i<ROWS;i++)
        for(j=0;j<COLUMN;j++)
            arr[i][j]=rand()%100;
}
```

מיון בועות למערך דו ממדי


```
void printArray(int arr[][COLUMN])
{
    int i, j;
    for(i=0;i<ROWS;i++)
    {
        for(j=0;j<COLUMN;j++)
            printf("%5d",arr[i][j]);
        printf("\n");
    }
}
```

מיון בועות למערך דו ממדי


```
void bubblesort(int data[][COLUMN])
{
        int i,j,k,temp;
        for(k = 0; k < ROWS * COLUMN; k ++)
        {
            for(i = 0; i < ROWS; i ++)
            {
                for(j = 0; j < COLUMN; j ++)
                {
```

מיון בועות למערך דו ממדי


```
if((data[i][j] > data[i][j + 1]) && j != COLUMN - 1)
{
    temp = data[i][j];
    data[i][j] = data[i][j + 1];
    data[i][j + 1] = temp;
}
if((data[i + 1][0] < data[i][j]) && (i != ROWS - 1))
{
    temp = data[i + 1][0];
    data[i + 1][0] = data[i][j];
    data[i][j] = temp;
}}}}}
```

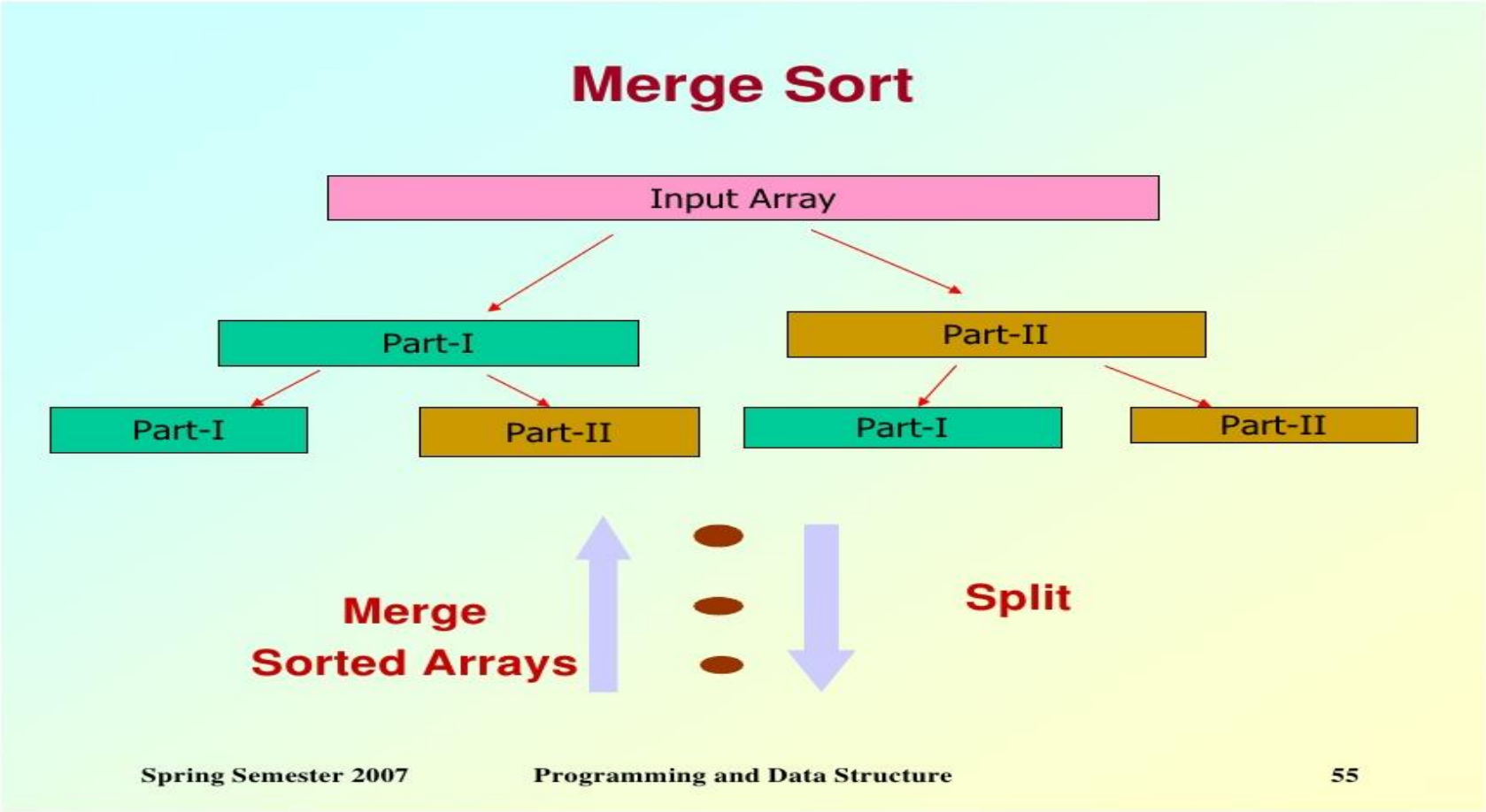
שאלות?

מיון מיזוג של מערך חד ממדי

- אלגוריתם מיון רגיל המבוסס על חלוקת המערך המקורי לשני מערכים, מיונם במיון בועות ומיזוגם בחזרה למערך אחד ממוין.
- מיון מיזוג (Merge Sort)
- אלגוריתם רגיל קל מאד למימוש ומתבסס על כך שקל מאוד למזג שני מערכים ממוינים אשר מוינו מראש במיון בועות:

1. מלא מערך חד ממדי במספרים אקראיים
2. חלק את המערך לשני מערכים
3. מיין כל אחד מהמערכים במיון בועות רגיל
4. מזג את שני המערכים הממוינים למערך אחד ממוין.

מיון מיזוג של מערך חד ממדי



דוגמה לתוכנית מיון מיזוג רגיל

הפונקציה printarr מדפיסה את המערך הגדול:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define SIZE 10
void printarr(int numbers[])
{
    int i;
    for(i = 0; i < SIZE; i++)
    printf("Array place %d the value is %d\n",i,numbers[i]);
    puts("\n");
}
```

דוגמה לתוכנית מיון מיזוג רגיל

הפונקציה printarr12 מדפיסה את המערכים הקטנים(חצי מהמערך הגדול):

```
void printarr12(int numbers[])
{
   int i;
   for(i = 0; i < SIZE/2; i++)
    printf(“In place %d the value is %d\n",i,numbers[i]);
   puts("\n");
}
```

דוגמה לתוכנית מיון מיזוג רגיל

הפונקציה initRandomValues ממלאת את המערך במספרים אקראיים:

```
void initRandomValues(int numbers[])
{
    int i;
    for(i=0;i<SIZE;i++)
        numbers[i]=rand()%100;
    puts("\tarr array:");
    printarr(numbers);
}
```

דוגמה לתוכנית מיון מיזוג רגיל

הפונקציה divarr מחלקת את המערך המקורי לשני מערכים קטנים:

```
void divarr(int num[],int num1[],int num2[])
{
   int i,j=0;
   for(i=0;i<SIZE/2;i++)  num1[i]=num[j++];
   puts("\tarr1 array:");
   printarr12(num1);
   for(i=0;i<SIZE/2;i++) num2[i]=num[j++];
   puts("\tarr2 array:");
   printarr12(num2);
}
```

דוגמה לתוכנית מיון מיזוג רגיל

הפונקציה sort ממיינת מיון בועות למערך הנשלח ונקלט אצלה:

```
void sort(int numbers[ ])
{
   int i, j=0,K, temp;
   for (i = (SIZE-1)/2; i >= 0; i--)
         for (j = 1; j <= i; j++)
                  if (numbers[j-1] > numbers[j])
                  {
                           temp = numbers[j-1];
                           numbers[j-1] = numbers[j];
                           numbers[j] = temp;
                  }
}
```

דוגמה לתוכנית מיון מיזוג רגיל

הפונקציה mergesort ממזגת שני מערכים ממוינים למערך אחד ממוין:

```
void mergesort(int sorted[], int arr1[], int arr2[])
{
  int i,j,k,m,n;
  m = n = SIZE/2;
  j = k = 0;
  for (i = 0; i < m + n;)
        if (j < m && k < n)
        {
            if (arr1[j] < arr2[k])
           {
                sorted[i] = arr1[j];        j++;
            }
```

דוגמה לתוכנית מיון מיזוג רגיל

הפונקציה mergesort ממזגת שני מערכים ממוינים למערך אחד ממוין:

```
        else
        {
                sorted[i] = arr2[k];    k++;
        }
  i++;    }
else
        if (j == m)
        {
                for (; i < m + n;)
                {
                        sorted[i] = arr2[k]; k++;       i++;
                }  }
```

דוגמה לתוכנית מיון מיזוג רגיל

הפונקציה mergesort ממזגת שני מערכים ממוינים למערך אחד ממוין:

```
else
{
        for (; i < m + n;)
        {
                sorted[i] = arr1[j];
                j++;
                i++;
        }
}
}
```

דוגמה לתוכנית מיון מיזוג רגיל

הפונקציה main התוכנית הראשית המפעילה את כל הפונקציות:

```
int main()
{
   int arr[SIZE]={0},arr1[SIZE/2]={0},arr2[SIZE/2]={0};
   int i;
   srand(time(NULL));
   initRandomValues(arr);
   divarr(arr,arr1,arr2);
   sort(arr1);
   printf("\tSorted arr1 array:\n");
   printarr12(arr1);
```

דוגמה לתוכנית מיון מיזוג רגיל

הפונקציה main התוכנית הראשית המפעילה את כל הפונקציות:

```
    sort(arr2);
    printf("\tSorted arr2 array:\n");
    printarr12(arr2);
    mergesort(arr,arr1,arr2);
    printf("\tSorted arr array:\n");
    printarr(arr);
    return 0;
}
```

שאלות?

מיון מיזוג רקורסיבי של מערך חד ממדי

- **מיון מיזוג (Merge Sort)**
- **אלגוריתם רקורסיבי קל למימוש ומתבסס על כך שקל מאוד למזג שני מערכים ממוינים:**

1. **מיין-מזג n איברים**
2. **אם n=1 (המערך של איבר אחד ממילא ממוין) חזור**
3. **מיין-מזג את n/2 האברים הראשונים**
4. **מיין-מזג את n/2 האברים האחרונים**
5. **מזג את 2 תוצאות המיון**

מיון מיזוג רקורסיבי של מערך חד ממדי

- אלגוריתם זה ממחיש היטב את הרעיון שבכדי להבהיר לעצמנו שהאלגוריתם רקורסיבי עובד, אין צורך "לראות בעיני רוחנו" מה בדיוק מתרחש בעת הביצוע, אלא מספיק לוודא כי:
- מתקיימת נכונות של שני חלקי הרקורסיה (צעד המעבר ומקרה הבסיס).
- כל שרשרת קריאות רקורסיביות אכן מסתיימת באחד ממקרי הבסיס

דוגמה לתוכנית מיון מיזוג רקורסיבי

הפונקציה merge ממזגת שני מערכים ממוינים:

```
#include <stdio.h>
#define MAX_SIZE 10
void merge(int arr[], int first, int mid, int last)
{
   int tmp_arr[MAX_SIZE] , int i=first ,  j=mid  ,k=0;
   while (i < mid && j < last)
   {
      if (arr[i]<arr[j])
      tmp_arr[k]=arr[i++];
      else
      tmp_arr[k]=arr[j++];
      ++k;
   }
```

דוגמה לתוכנית מיון מיזוג רקורסיבי

המשך הפונקציה merge שממזגת שני מערכים ממוינים:

```
if (j < last)
        for ( ; j<last;++j)
                tmp_arr[k++]=arr[j];
else
        for ( ; i<mid;++i)
                tmp_arr[k++]=arr[i];
for (k=first;k<last;++k)
        arr[k]=tmp_arr[k-first];
}
```

דוגמה לתוכנית מיון מיזוג רקורסיבי

הפונקציה mergesort הממיינת את המערך הממוין:

```
void mergesort(int arr[], int first, int last)
{
        int i, mid;
        if ( first<last )
        {
        for (i=0;i<first;++i)
                printf(" ");
        printf("Sorting from %d to %d\n", first, last);
        mid=(last+first)/2;
```

דוגמה לתוכנית מיון מיזוג רקורסיבי

המשך פונקציה mergesort הממיינת את המערך הממוין:

```
    if (mid==first)
        ++mid;
    mergesort(arr,first, mid-1);
    mergesort(arr,mid, last);
    merge(arr,first,mid, last+1);
    for (i=0;i<first;++i)
        printf(" ");
    printf("Merging from %d to %d\n", first, last);
    }
}
```

דוגמה לתוכנית מיון מיזוג רקורסיבי

הפונקציה printArr המדפיסה את המערך:

```
void printArr(int arr[], int length)
{
        int i;
        for (i=0;i<length;++i)
                printf("%3d ",arr[i]);
        printf("\n\n");
}
```

דוגמה לתוכנית מיון מיזוג רקורסיבי

התוכנית הראשית:

```
void main ()
{
    int  length, arr[SIZE] =  {20,1,0,-10,89,-22,76,31,8};
    length = MAX_SIZE-1;
    puts("Original Array:");
    printArr(arr, length);
    mergesort(arr,0, length-1);
    puts("\nSorted Array:");
    printArr(arr, length);
}
```

דוגמה נוספת למיון מיזוג רקורסיבי

- אלגוריתם מיון רקורסיבי – מיון מיזוג merge sort
- אם נתונים שני מערכים ממוינים קל למזג אותם למערך אחד ממוין: עוברים על שניהם בו-זמנית ובכל פעם שנתקלים באבר במערך אחד הקטן מהאבר עליו נמצאים במערך השני מעתיקים את האבר הזה למערך שלישי ומתקדמים באותו מערך.
- לדוגמה: נתונים המערכים:

A	4	7	9	16	27	78	89	90	100

B	3	5	6	30	45	55	103

מיון מיזוג רקורסיבי של מערך חד ממדי

- בהתחלה נעתיק את 3 ונתקדם במערך B, אח"כ את 4 ונתקדם במערך A. אח"כ את 5 ונתקדם ב B ואת 6 ושוב נתקדם ב B, וכן הלאה.

A	4	7	9	16	27	78	89	90	100

B	3	5	6	30	45	55	103

- התוצאה:

AB	3	4	5	6	7	9	16	27	30	45	55	78	89	90	100	103

מיון מיזוג רקורסיבי של מערך חד ממדי

- זהו הבסיס לאלגוריתם המיון הרקורסיבי הנקרא merge sort:
- נניח שאנו רוצים למיין מערך של מספרים. נחלק את המערך לשני מערכים בגודל זהה (או כמעט זהה, אם גודל המערך אי-זוגי), נמיין כל חלק בנפרד ונמזג את שני המערכים הממוינים למערך אחד ממוין.

מיון מיזוג רקורסיבי של מערך חד ממדי

- אלא שאת כל אחד מהחלקים נמיין בעזרת אותו אלגוריתם עצמו, כלומר, נחלק אותו לשני חלקים, נמיין כל אחד בנפרד ונמזג.
- גם את כל אחד מהמערכים הקטנים (בגודל רבע מהמערך המקורי) נמיין בעזרת אותו אלגוריתם, וכן הלאה... הרקורסיה תיעצר ברגע שנצטרך למיין מערכים בגודל 1.
- נמזג כל שני מערכים בגודל 1 למערך ממוין בגודל 2, ונמשיך למזג, עד שנקבל את המערך המקורי ממוין.

פונקציית המיון הרקורסיבי

הפונקציה merge ממזגת שני מערכים ממוינים:

```
void mergeArray(int a[], int na, int b[], int nb)
{
    int *temp, i=0, ai=0, bi=0;
    temp = (int *)malloc((na+nb)*sizeof(int));
    if(temp == NULL)
    {
    printf("not enough memory to complete mission.\n");
    exit(1);
    }
```

פונקציית המיון הרקורסיבי

המשך פונקציה merge הממזגת שני מערכים ממוינים:

```
while(ai < na && bi < nb)
{
        if(a[ai] < b[bi])
                temp[i++] = a[ai++];
        else
                temp[i++] = b[bi++];
}
```

פונקציית המיון הרקורסיבי

המשך פונקציה merge הממזגת שני מערכים ממוינים:

```
        while(ai < na)
                temp[i++] = a[ai++];
        while(bi < nb)
                temp[i++] = b[bi++];
        // copy temp to a, and free allocated memory
        memcpy(a,temp,(na+nb)*sizeof(int));
        // in <memory.h>
        free(temp);
}
```

פונקציית המיון הרקורסיבי

המערכים a ו- b ממוזגים לתוך המערך temp. כאשר הגענו לסוף אחד המערכים. האיברים הנותרים במערך השני מועתקים ל temp. התוצאה מועתקת חזרה לתוך המערך a ו- temp משוחרר.

הפונקציה mergeSort:

```
void mergeSort(int *left, int *right)
{//beginning at left and ending at right using merge sort recursively
        int *mid = left + (right - left)/2;
        if(left >= right)          // if less than 2 elements do nothing
                return;
        mergeSort(left, mid);
        mergeSort(mid+1, right);
        mergeArray(left, mid - left +1, mid + 1, right - mid);
}
```

פונקציית מיון מיזוג רקורסיבית

- הפונקציה מקבלת שתי כתובות, left ו right, וממיינת את המספרים השמורים בכתובות שביניהן, כולל שתי הכתובות הקיצוניות. האלגוריתם עובד כך שכל המיון מתבצע בתוך המערך המקורי.
- ניתן לכתוב את הפונקציה גם בכתיב של מערכים במקום כתיב של פוינטרים.

פונקציית מיון מיזוג רקורסיבית


```
void mergeSort2(int a[], int size)
{
        int half=size/2;
        if (size<=1)
                return;
        mergeSort2(a, half);
        mergeSort2(a+half, size-half);
        mergeArray(a, half, a+half, size-half);
}
```

פונקציית מילוי המערך במספרים

הפונקציה משתמשת בפונקציות rand_fill_array הממלאת מערך במספרים אקראיים, ו print_array המדפיסה מערך:

```
void rand_fill_array(int *a, int size, int module)
{
        int i;
        srand(time(NULL));
        for (i=0; i< size; i++)
                a[i] = rand()%module;
}
```

פונקציית הדפסת המערך


```
void print_array(int *a, int n)
{
        int i;
        for (i = 0; i < n; ++i)
        {
                printf("%7d", a[i]);
                if ((i+1)%10 == 0)
                        putchar('\n');
        }
        putchar('\n');
}
```

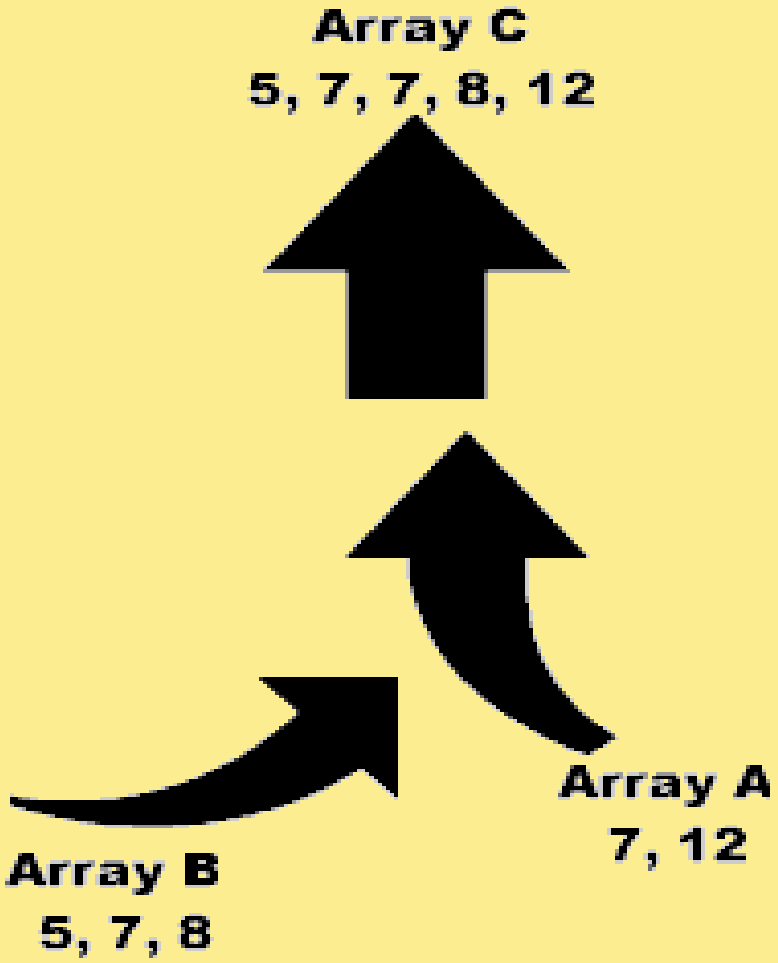
פונקציית הפונקציה quick_sort:


```
//recursive function that works on the low part and the high part and sorts them by the function partition
void quick_sort(int a[],int n)
{
        int i;
        if(n<=1)    //till the array is of size 1
                return;
        i=partition(a,n);
        quick_sort(a,i);     //recursion on the low part
        quick_sort(a+i+1,n-i-1);
        //recursion on the high part
}
```

פונקציית main:


```
int main()
{
        int *numbers, array_size, module = 1000;
        printf("Enter amount of numbers to sort: ");
        scanf("%d", &array_size);
        numbers = (int *)malloc(array_size * sizeof(int));
        rand_fill_array(numbers, array_size, module);
        mergeSort(numbers, numbers + array_size - 1);
        printf("Sorted array:");
        print_array(numbers, array_size);
        free(numbers);
        return 0;
}
```

סיכום



- הצורך במיון בועות
- הצורך בחיפוש לינארי
- הצורך בחיפוש בינארי
- הצורך במיון מערך דו ממדי
- הצורך במיון מיזוג רקורסיבי

שאלות?