מרתון מבני נתונים

הערות:

* בפסאודו קוד, לרוב נתחיל את המערכים מ 1 עד n (כולל)
* סיבוכיות (חישוב סיבוכיות זמן ריצה)
  + פונקציות רקורסיביות
* מיונים
  + מיון מיזוג
  + מיון מהיר
  + מיון בסיס
  + מיון מניה
  + חסם תחתון על סיבוכיות
* מיזוג
* חיפוש בינארי
* מחסנית
* תור
* עצים בינאריים
  + הגדרות
  + סריקה
  + עץ חיפוש
  + עץ AVL

**סיבוכיות זמן ריצה**

* כל הסיבוכיות נמדדת בסדרי גודל ולא בדיוק.
* הכל תלוי בקלט, ככל שהקלט גדל, בכמה זמן הריצה ייגדל?

מי משפיע? לולאות, פונקציות רקורסיביות.

דוגמאות:

1. מה סיבוכיות הפונקציה הבאה:

f(int[] arr)

n = arr.length

for( i=1; i <= n; i +=2)

print(arr[i]

סיבוכיות: . כי פעולת ההדפסה תתבצע פעמים.

1. מה סיבוכיות הפונקציה הבאה:

f(int[] arr)

n = arr.length

for( i=1; i <= n; i +=2)

for(j = 1; j <=n; j+=5)

print(“hi”)

for(i =1; i<=n; i\*=10)

print(“hi”)

סיבוכיות: 2 לולאות ראשונות: בכל איטרציה של הלולאה החיצונה, הלולאה הפנימית תבצע

פעולות. הלולאה החיצונה תתבצע פעמים ולכן סה"כ יבוצעו: ולכן סיבוכיות 2 הלולאות תהיה .

הלולאה השלישית: - התנאי ליציאה מהלולאה, כאשר: אחרי פעמים.  
 נבודד את k ונקבל: ולכן סיבוכיות הלולאה השלישית היא: .

סה"כ סיבוכיות הפונקציה: .

1. מה סיבוכיות הפונקציה הבאה:

f(int n)

for( i=n; i >= 1; i /=2)

print(“hi”)

if(n>1)

for(j=1; j<=4; j++)

f(n/2)

סיבוכיות: נסמן ב את זמן הריצה של f עבור הקלט n.

ולכן: , .

מפתחים את הפונקציה הרקורסיבית:

ומכאן, לאחר k צעדים נקבל:

. נדרוש ש: . נבודד את k ונקבל: .

נקבל:

**חיפוש בינארי**

בהינתן מערך ממויין, מספר x.   
המטרה למצוא את x בתוך המערך.

מוצאים את אמצע המרך, ובודקים האם x באמצע, אם כן - מצאנו. אם לא - אם x גדול ממשיכים לחפש בצד ימין של המערך, אם x קטן אז ממשיכים לחפש בצד שמאל. חוזרים על התהליך שוב ושוב, כל פעם על מערך קטן יותר עד שמוצאים את x או עד שמגיעים למערך בגודל 1 ואם זה לא שם אז x לא נמצא.

binarySearch(A,x) , |A|=n

low = 1, high = n

while(low <= high)

mid = (low + high)/2

if(A[mid] == x) return mid

else if(A[mid] < x) low = mid+1

else high = mid-1

return -1

סיבוכיות: כי בכל פעם חוצים את המערך ל 2 ומחפשים בצד אחד עם שמגיעים למערך בגודל 1.

מכיוון שבכל פעם גודל המערך מתחלק ב 2, תוך צעדים לכל היותר נסיים את החיפוש.

מימוש רקורסיבי:

binarySearch(A,x) , |A|=n

return binarySearch(A,x,1,n)

binarySearch(A,x,low,high)

if(high < low) return -1

else

mid = (low + high)/2

if(A[mid] == x) return mid

else if(A[mid] < x) return binarySearch(A,x,mid+1,high)

else return binarySearch(A,x,low,mid-1)

סיבוכיות: נסמן ב את זמן הריצה (של הפונקציה השנייה).

מכאן:

פיתוח:

**מיזוג**

בהינתן 2 מערכים ממויינים, יש למזגם למערך אחד ממויין.

רעיון: שמים מצביע לאיבר הראשון בכל מערך (סה"כ 2 מצביעים)

מי שקטן יותר מבין 2 האיברים יהיה האיבר הכי קטן במערך התשובה.  
מקדמים את המצביע של מי שהיה קטן יותר לאיבר הבא במערך שלו.  
חוזרים על אותו תהליך שוב ושוב עד שאחד המערכים מסתיים ואז מכניסים את כל מה שנשאר מהמערך השני.

merge(A,B), |A|=n, |B|=m

C = new Array[n+m]

i = 1, j = 1, k =1

while(i <=n and j <= m)

if(A[i] < B[j])

C[k] = A[i]

i++

else

C[k] = B[j]

j++

k++

while(i <= n)

C[k] = A[i]

i++

k++

while(j <=m)

C[k] = B[j]

j++

k++

return C

סיבוכיות: .

**מיונים**

חסם תחתון על מיון מבוסס השוואות (כאשר לא ידוע שום מידע על איברי המערך ומשתמשים באי שוויונות כדי להחליט מי יותר גדול ומי יותר קטן)

החסם התחתון הוא: כאשר הוא גודל המערך.

הוכחה: אם ניקח מערך בגודל n. אז יש מערכים שונים שניתן ליצור מאותם איברים של המערך שלקחנו.

לדוגמא: אז ניתן ליצור ממנו: .

כל אלגוריתם מבוסס השוואות, לוקח בכל פעם 2 איברים (מקומות במערך), בודק מי גדול ממי ובהתאם לתשובה פועל אחרת. לכן, אם נסתכל על עץ ההחלטות, זהו עץ בינארי.

מכיוון שיש אופציות שונות, ולכל 2 אופציות שניקח תהיה השוואה אחת לפחות בה התשובה תהיה שונה.

לכן בעץ ההחלטות יש לפחות עלים.

גובהו של עץ בינארי עם עלים הוא לפחות .

לכן יש בעץ ההחלטות יש גובה של לפחות .

בכל פעולה (שאלה והחלטה) יורדים צעד אחד בעץ ולכן כמות הפעולות במקרה הגרוע היא תהיה לאורך גובה העץ שהוא .

כעת,

ומצד שני:

ולכן כל אלגוריתם מבוסס השוואות יבצע לפחות פעולות.

מטרת כל המיונים: בהינתן מערך - למיין אותו מהקטן לגדול.

1. מיון מיזוג:

בהינתן מערך, חלק אותו ל 2, מיין בנפרד כל חלק (קיבלנו כאילו 2 מערכים ממויינים כל אחד בנפרד) ואז מזג אותם (עם האלגוריתם למיזוג).

איך ממיינים כל חלק? רקורסיבית - מחלקים גם אותם ל 2 חלקים כל אחד, ממיינים וממזגים.

תנאי עצירה - מערך בגודל 1 - כי הוא כבר ממויין.

mergeSort(A)

mergeSort(A,1,n)

mergeSort(A,low,high)

if(low == high) return

else

mid = (low + high)/2

mergeSort(A,low,mid)

mergeSort(A,mid+1,high)

B[] = A[low...mid] // copy A from index low to mid into temp array B

C[] = A[mid+1...high] // copy A from index mid+1 to high into array C

D = merge(B,C)

A[low...high] = D // copy D from 1 to end of D into A from low to high

סיבוכיות:   
אם הוא זמן הריצה עבור מערך בגודל n. אז:

פיתוח:

2. מיון מהיר:

רעיון: בוחרים איבר (שנקרא איבר הציר), בדרך כלל בוחרים את הראשון.  
עוברים על כל המערך ושמים את כל האיברים הגדולים מאיבר הציר מימינו ואת הקטנים משמאלו. (בין האיברים עצמם ייתכן שאין סדר)

לאחר המעבר הזה, איבר הציר נמצא בהכרח במקומו הממויין.  
רקורסיבית ממיינים כל צד בנפרד. עד שמגיעים למערך בגודל 1.

QuickSort(A)

QuickSort(A,1,n)

QuickSort(A,low,high)

if(low == high) return

else

pivotIndex = Pivot(A,low, high)

QuickSort(A,low, pivotIndex-1)

QuickSort(A, pivotIndex+1, high)

Pivot(A,low,high)

p = A[low]

i = low+1, j = high

while(i < j)

if(A[j] > p) j--

else if(A[i] < p) i++

else swap(A,i,j)

swap(A,low,j)

return j

סיבוכיות: מקרה ממוצע: . מקרה גרוע: .

נסמן ב את זמן הריצה. אז:

בממוצע בסיכוי יותר גבוה שהאיבר הראשון הוא מישהו באמצע או קרוב לאמצע ולכן הסיבוכיות תהיה כמו שהיא .

במקרה הגרוע, אם נקבל חלק אחד בגודל 1 והחלק השני הוא כל השאר. (ייתקבל כשהמערך ממויין)

כלומר:

נפתח:

3. מיון מניה:  
נתון שאיברי המערך נמצאים בתחום ונסמן: (גודל הטווח)

רעיון: ניצור מערך עזר בגודל .  
נעבור על המערך המקורי, ואת האיבר שנמצא במיקום ה נוסיף 1 במערך החדש במיקום x.

לדוגמא: אם המערך המקורי הוא: ונתון שהטווח הוא בין 1 ל 10

אז ניצור מערך תדירויות חדש בגודל 10: .   
נעבור על המערך המקורי, עבור 9 - נוסיף 1 למיקום 9 במערך החדש, עבור 6 - נוסיף 1 למיקום 6. וכך הלאה ונקבל: .

כעת, נעבור על מערך התדירויות לפי הסדר, ולפי המיקום, נמיר בחזרה לאיבר המקורי ונשים אותו במערך המקורי החל מההתחלה לפי הכמות הנתונה במערך התדירויות.  
בדוגמא נקבל: .

אם הטווח לא מתחיל מ 1 (בפסודו קוד) או מ 0 (ב java) אז איך מזיזים את הטווח?  
לדוגמא, אם הטווח מתחיל ממינוס 3 אז אין במערך מיקום מינוס 3.

תשובה: נבצע הזזה, את האיבר x נכניס למיקום x-min (ב java).

בפסאודו קוד כאן המערך יתחיל מ 0.

countingSort(A), |A|=n

min = min(A)

max = max(A)

freq[] = new array[max - min + 1]

for i = 0 to n-1

freq[A[i] - min]++

k = 0

for i = 0 to freq.length-1

for j = 1 to freq[j]

A[k] = j + min

k++

סיבוכיות: כאשר הוא גודל המערך, הוא הטווח.

4. מיון בסיס:

עבור מספרים שלמים חיוביים בדרך כלל, כאשר ידוע מספר הספרות המקסימאלי. נסמנו ב d.

מתחילים למיין כמו מיון מנייה לפי ספרת האחדות בלבד. לאחר מכן, לפי ספרת העשרות, ואז המאות, וכן הלאה. אם לדוגמא מסתכלים על ספרת המאות של מספר עם 2 ספרות אז היא 0.

radixSort(A) , |A|=n

max = max(A)

d = (int)+ 1

for i = 0 to d-1

T[] = new Array[n]  
 t = 0

for j = 0 to 9

for k = 0 to n-1

if(((A[k] / ) mod 10) == j)

T[t] = A[k]

t++

A = T

סיבוכיות: .