האוניברסיטה הפתוחה

מבני נתונים ומבוא לאלגוריתמים
מפגש הנחיה מס' 12

מדעי המחשב, קורס מס' 20407

סמסטר 2016ב

מנחה: ג'ון מרברג

מה ראינו במפגש הקודם?

- הרחבה של מבני נתונים (המשך)
  - הרחבה – הכללה של תחזוקת שדה מרחיב
  - דוגמאות נוספות להרחבה של עץ אדום שחור
- חזרה (חלק ראשון)
  - הגדרת מבנה נתונים מופשט (ADT)
    - הפוטנציאל של כל אחד ממבני הנתונים הבסיסיים
    - שילוב מספר מבני נתונים בסיסיים לצורך הגדרת ADT

האוניברסיטה הפתוחה

מפגש שנים-עשר (אחרון)

- נושא השיעור
  - טיפים לעבודה בבחינה
- חזרה (המשך)
  - סימונים אסימפטוטים
  - נוסחאות נסיגה
    - שיטת האיטרציה
    - שיטת האב
  - מיון, חיפוש, בחירה, דירוג
  - תרגילים ככל שנספיק

מבוסס על מצגת של ברוך חייקין ואיציק בייז

טיפים לעבודה בבחינה

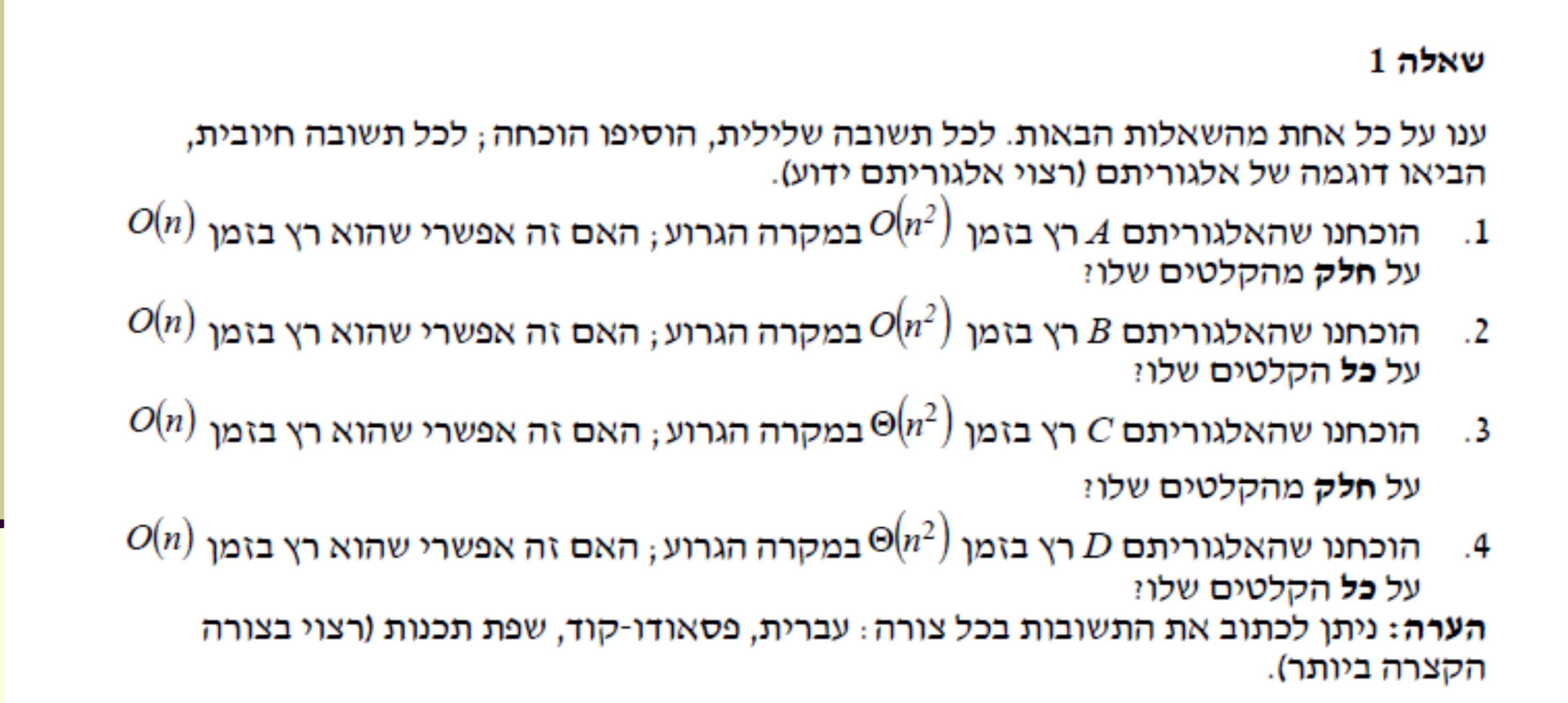
- כדאי לעבור תחילה על כל הבחינה – כל השאלות
- ניתן לענות על השאלות בסדר כלשהו
  - התחילו עם השאלה הנוחה לכם
  - אל "תתקעו" בשאלה שאין לכם יכולת להתקדם בה – חזרו אליה מאוחר יותר
- כשנגשים לפתור שאלה, קודם יש לקרוא היטב ולהבין את השאלה
  - הרבה טעויות נובעות מחוסר הבנה של השאלה
  - אם אפשר, קחו דוגמא כדי לבחון את אופי הבעיה, או להבין כיצד האלגוריתם הנתון או הנדרש מתנהג
- ניתן לצטט טענות ונוסחאות מחומר הלימוד ללא הוכחה
  - חשוב לציין מהיכן נלקחו – למען הסר ספק
- מומלץ לבצע סימולציה מלאה של בחינה בתנאים אמיתיים
  - זמן מוקצב, חומר כתוב בלבד, ללא מחשב ואינטרנט

טיפים לעבודה בבחינה (המשך)

- הצגה של אלגוריתם
  - יש לתת תמיד הסבר מילולי כללי (קצר) לכל אלגוריתם שאתם נדרשים להציג
  - אם נדרשתם לכתוב את האלגוריתם
    - כתבו בפסודו-קוד באופן ברור ומדויק, עם סימונים עקביים, אינדנטציה
    - מומלץ להיעזר במספרי שורות בהסברים והוכחות
    - אין צורך להעתיק תוכן של אלגוריתמים ידועים, אלא רק לקרוא להם
  - אם נדרשתם לתאר את האלגוריתם
    - תנו תיאור מובנה וממצה בשפה חופשית
    - ברמת פירוט סבירה בהתאם לבעיה
  - יש להראות תמיד את נכונות האלגוריתם, כלומר לטעון מדוע הוא נותן פתרון לבעיה
    - עם או בלי הוכחה פורמאלית – לפי הנדרש ולפי שיקולכם
  - יש לנתח תמיד את זמן הריצה האסימפטוטי

האוניברסיטה הפתוחה

תרגיל: סימונים אסימפטוטים



שאלה 1

ענו על כל אחת מהשאלות הבאות. לכל תשובה שלילית, הוסיפו הוכחה; לכל תשובה חיובית, הביאו דוגמה של אלגוריתם (רצוי אלגוריתם ידוע).

1. הוכחנו שהאלגוריתם A רץ בזמן $O(n^2)$ במקרה הגרוע; האם זה אפשרי שהוא רץ בזמן $O(n)$ על **חלק** מהקלטים שלו?
2. הוכחנו שהאלגוריתם B רץ בזמן $O(n^2)$ במקרה הגרוע; האם זה אפשרי שהוא רץ בזמן $O(n)$ על **כל** הקלטים שלו?
3. הוכחנו שהאלגוריתם C רץ בזמן $\Theta(n^2)$ במקרה הגרוע; האם זה אפשרי שהוא רץ בזמן $O(n)$ על **חלק** מהקלטים שלו?
4. הוכחנו שהאלגוריתם D רץ בזמן $\Theta(n^2)$ במקרה הגרוע; האם זה אפשרי שהוא רץ בזמן $O(n)$ על **כל** הקלטים שלו?

הערה: ניתן לכתוב את התשובות בכל צורה: עברית, פסאודו-קוד, שפת תכנות (רצוי בצורה הקצרה ביותר).

שיטת האב – הגדרה פורמאלית

- שיטת האב מתייחסת לנוסחת הנסיגה: $T(n)=aT(\frac{n}{b})+f(n)$ כאשר $b>1$, $a\geq 1$
- השיטה מבחינה בין שלשה מקרים של עלות העבודה בעץ הקריאות:

1. **הגורם הדומיננטי הוא עלות העבודה המתבצעת בעלים (= מספר העלים)**

   אם קיים קבוע $\varepsilon > 0$ כך ש- $f(n)=O(\frac{n^{\log_b a}}{n^{\varepsilon}})$ אז $T(n)=\Theta(n^{\log_b a})$

2. **העבודה מתחלקת באופן שווה בין כל הרמות בעץ**

   אם $f(n)=\Theta(n^{\log_b a})$ אז $T(n)=\Theta(n^{\log_b a}\log n)$

3. **הגורם הדומיננטי הוא עלות העבודה בשורש**

   אם קיים קבוע $\varepsilon > 0$ כך ש- $f(n)=\Omega(n^{\log_b a}n^{\varepsilon})$ אז $T(n)=\Theta(f(n))$

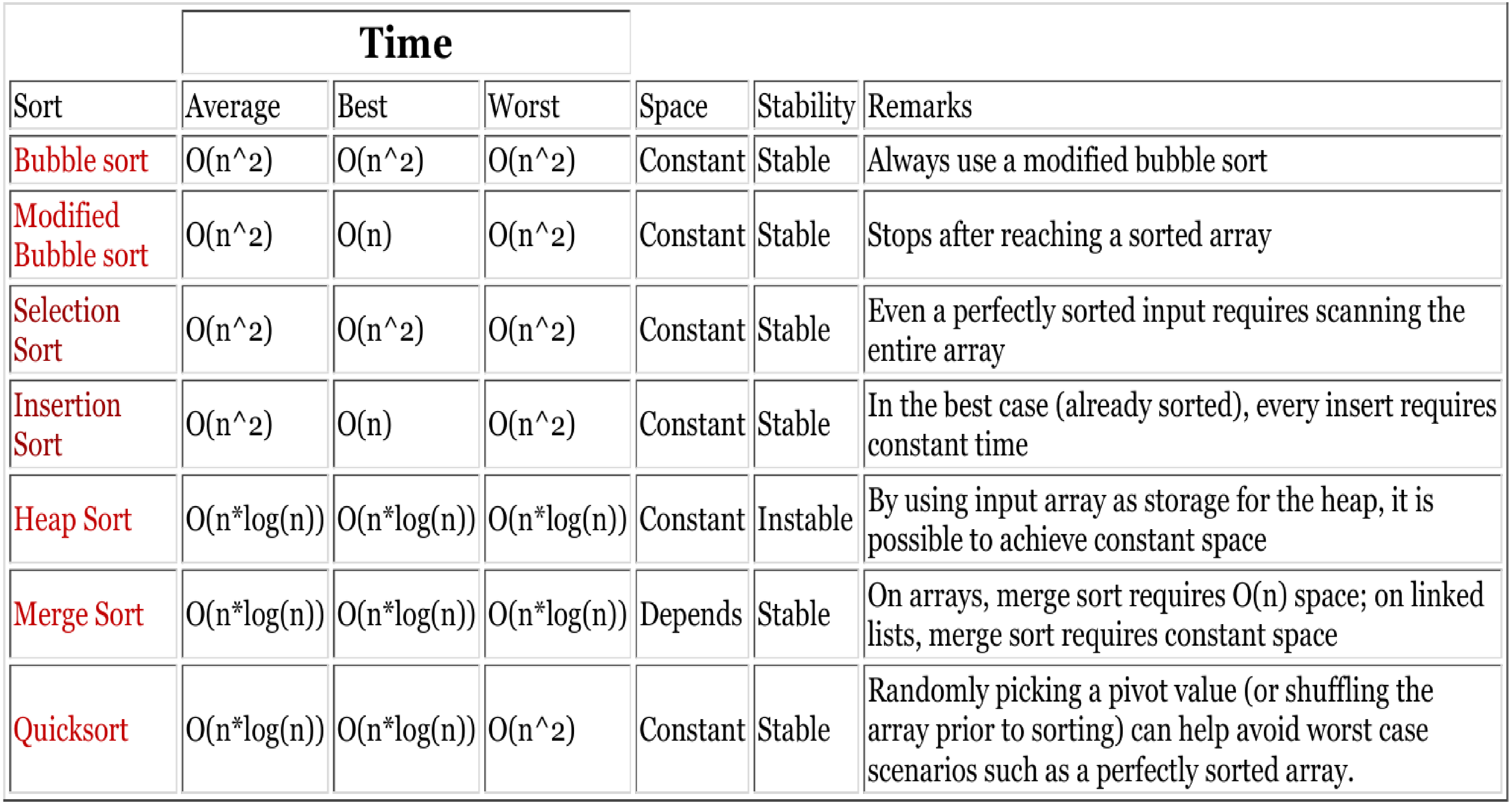
   במקרה 3 חייב להתקיים גם תנאי הרגולריות על *f(n)* כדלקמן:

   קיימים קבועים $c<1$ ו- n_0 כך ש- $af(\frac{n}{b})\leq cf(n)$ לכל $n\geq n_0$

(*) הערה לגבי היחס בין הגדלים במקרים 1, 3:

נשים לב שהגורם הדומיננטי חייב להיות גדול פולינומיאלית מהגורם השני, בפקטור של n^{ε}

מיון מבוסס השוואות

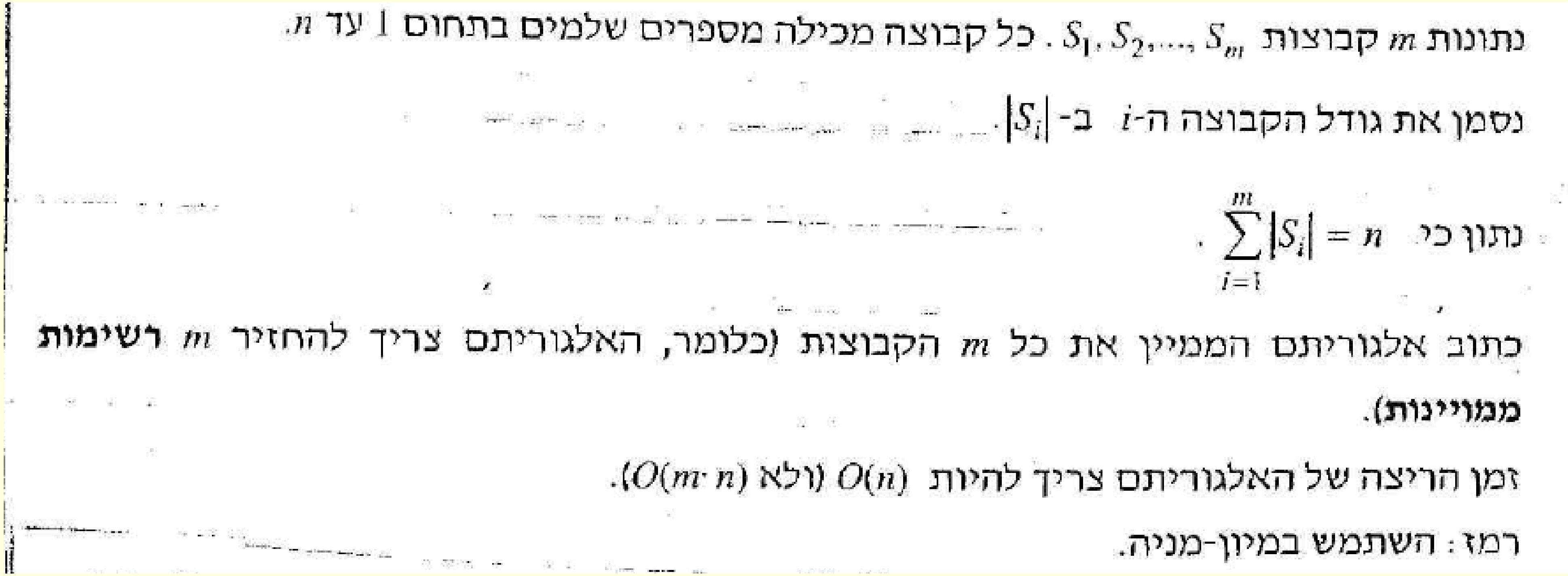


	Time					
Sort	Average	Best	Worst	Space	Stability	Remarks
Bubble sort	O(n^2)	O(n^2)	O(n^2)	Constant	Stable	Always use a modified bubble sort
Modified Bubble sort	O(n^2)	O(n)	O(n^2)	Constant	Stable	Stops after reaching a sorted array
Selection Sort	O(n^2)	O(n^2)	O(n^2)	Constant	Stable	Even a perfectly sorted input requires scanning the entire array
Insertion Sort	O(n^2)	O(n)	O(n^2)	Constant	Stable	In the best case (already sorted), every insert requires constant time
Heap Sort	O(n*log(n))	O(n*log(n))	O(n*log(n))	Constant	Instable	By using input array as storage for the heap, it is possible to achieve constant space
Merge Sort	O(n*log(n))	O(n*log(n))	O(n*log(n))	Depends	Stable	On arrays, merge sort requires O(n) space; on linked lists, merge sort requires constant space
Quicksort	O(n*log(n))	O(n*log(n))	O(n^2)	Constant	Stable	Randomly picking a pivot value (or shuffling the array prior to sorting) can help avoid worst case scenarios such as a perfectly sorted array.

חיפוש/בחירה/דירוג/בניה

- במערך
  - חיפוש לינארי $O(n)$
  - חיפוש בינארי $O(\log n)$ – במערך ממוין
  - מציאת מינימום/מקסימום/חציון $O(n)$
  - בחירת האיבר ה- i בסדר הממוין $O(n)$
  - דירוג איבר נתון $O(n)$
- בטבלת גיבוב
  - חיפוש $O(1)$ בממוצע
  - בניה $O(n)$ (בגיבוב עם שרשור)
- בערמה
  - חיפוש $O(n)$
  - מציאת מינימום או מקסימום $O(1)$ - אך לא שניהם
  - בניה $O(n)$
- בעץ אדום שחור
  - חיפוש $O(\log n)$
  - מציאת מינימום ומקסימום $O(\log n)$ ($O(1)$ עם תחזוקת שני משתנים)
  - בחירה ודירוג $O(\log n)$ – בעץ ערכי מיקום (עץ א"ש עם הרחבה)
  - בניה $O(n\log n)$

תרגיל: מיון מניה



נתונות m קבוצות $S_1, S_2, \ldots, S_m$. כל קבוצה מכילה מספרים שלמים בתחום 1 עד n.

נסמן את גודל הקבוצה ה-i ב- $|S_i|$.

נתון כי $\sum_{i=1}^{m} |S_i| = n$.

כתוב אלגוריתם הממיין את כל m הקבוצות (כלומר, האלגוריתם צריך להחזיר m **רשימות ממויינות**).

זמן הריצה של האלגוריתם צריך להיות $O(n)$ (ולא $O(m \cdot n)$).

רמז: השתמש במיון-מניה.

תרגיל: שכיחות של מפתחות

הציעו מבנה נתונים S התומך בפעולות הבאות (N מציין את מספר האיברים ב- S; n מציין את מספר המפתחות השונים זה מזה):

SEARCH (S,k): חיפוש אחר המפתח k במבנה S;

INSERT (S,k): הכנסת איבר חדש בעל המפתח k למבנה S;

DELETE (S,k): מחיקת איבר כלשהו בעל המפתח k מהמבנה S;

FREQUENCY (S,k): החזרת מספר האיברים בעלי המפתח k שבמבנה S;

SELECT (S,i): החזרת ערך המיקום ה- i של המבנה S (האיבר ה- i הקטן ביותר בין כל N האיברים של S).

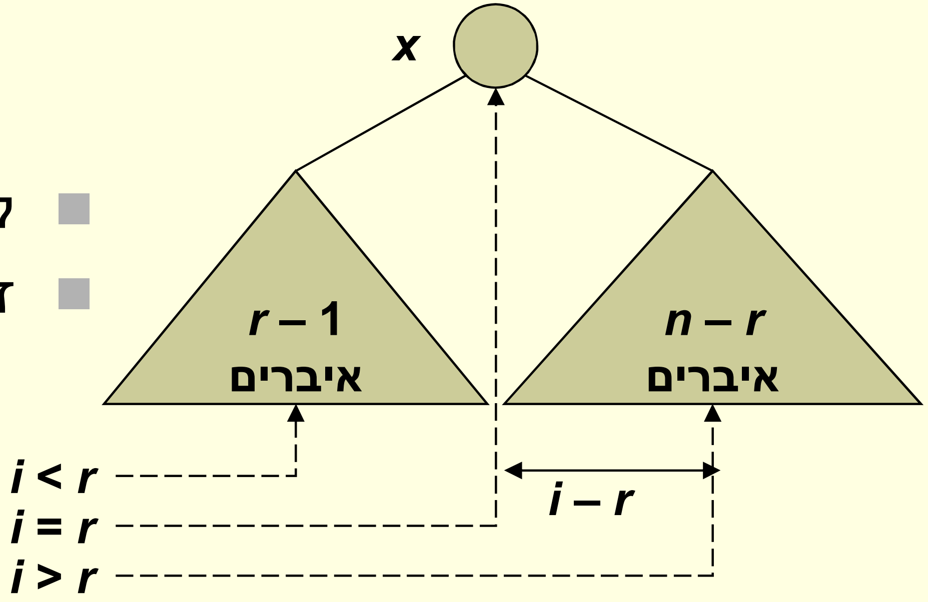
זמן הריצה הנדרש של כל אחת מהפעולות הינו $\Theta(\lg n)$.

חיפוש מפתח לפי דירוג נתון בעץ ערכי מיקום

- בהינתן צומת x ודירוג i, מצא בתת-העץ המושרש ב-x את הצומת שמפתחו מדורג במקום ה-i *בסדר הממוין של המפתחות* בתת-העץ
- הרעיון הוא להשוות את i לדירוג r של *שורש תת-העץ* x

```
OS-Select(x, i)
    r ← size[left[x]] + 1
    if i = r
        then return x
    if i < r
        then return OS-Select(left[x], i)
        else return OS-Select(right[x], i – r)
```

- קריאה חיצונית: **OS-Select**(*root*[*T*], *i*)
- זמן ריצה: $O(\lg n)$

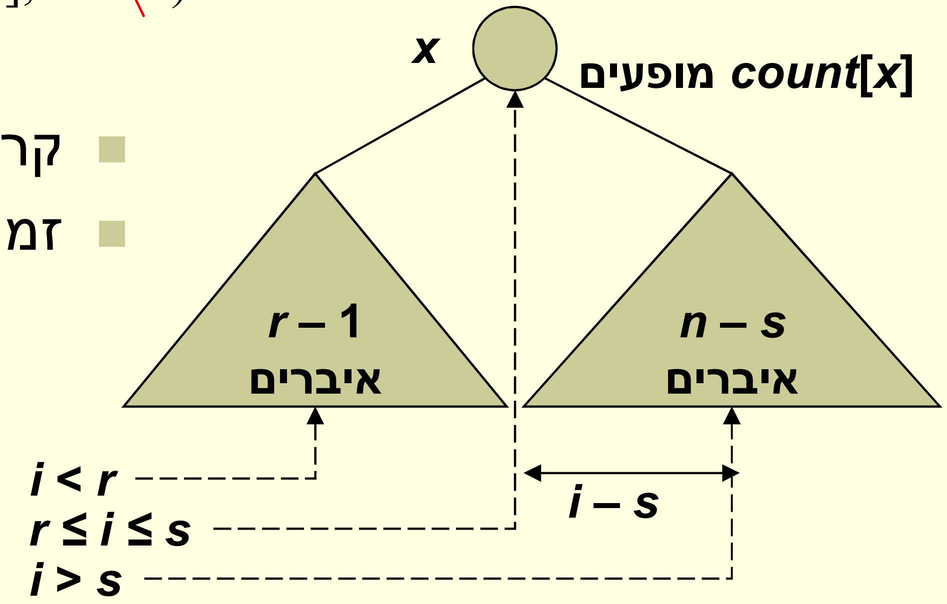


חיפוש מפתח לפי דירוג נתון בעץ שכיחויות

- בהינתן צומת *x* ודירוג *i*, מצא בתת-העץ המושרש ב-*x* את הצומת שמפתחו מדורג במקום ה-*i* *בסדר הממוין של המפתחות* בתת-העץ
- הרעיון הוא להשוות את *i* <u>לטווח הדירוגים</u> של *מופעי המפתח* של *x*

```
Freq-Select(x, i)
1.    r ← size[left[x]] + 1
1.5   s ← r + count[x] - 1
2.    if ~~i = r~~ r ≤ i ≤ s
3.        then return ~~x~~ head[x]
4.    if i < r
5.        then return Freq-Select(left[x], i)
6.        else return Freq-Select(right[x], i – ~~r~~ s)
```

- קריאה חיצונית: **Freq-Select**(*root*[*T*], *i*)
- זמן ריצה: $O(\lg n)$



קוד בכחול נכנס
קוד באדום יוצא

האוניברסיטה הפתוחה

תרגיל: מטריצה

שאלה 3

הציעו מבנה נתונים לתחזוקת מטריצה ריבועית $M[n,n]$. המבנה חייב לתמוך בפעולות הבאות בזמנים הנדרשים:

$INIT(M)$: אתחול המטריצה M (כל התאים מקבלים את הערך 0); זמן הריצה $\Theta(n^2)$;

$READ(M,i,j)$: החזרת הערך $M[i,j]$; זמן הריצה $O(1)$;

$UPDATE(M,i,j,v)$: העדכון $v \rightarrow M[i,j]$; זמן הריצה $O(1)$;

$TRANSPOSE(M)$: החלפת האיברים $M[j,i] \leftrightarrow M[i,j]$, לכל (i,j), $i < j$;

זמן הריצה: $O(1)$;

$UPGRADE(M,d)$: הוספת הערך d לכל איברי המטריצה M; זמן הריצה $O(1)$;

$SUM(M)$: החזרת סכום כל האיברים של M; זמן הריצה $O(1)$.