חישוב של אופרטורים רלציוניים

Evaluation of Relational Operators

1

חישוב יעיל של שאילתות

- שאילתות מתורגמות לביטוי אלגברי שאותו יש לחשב
- נדרש:
  - חישוב יעיל של כל אופרטור בביטוי
  - תוכנית ביצוע יעילה לחישוב הביטוי

2

יחסים לדוגמאות

- נתונים שני יחסים

Sailors(*sid*:integer, *sname*:string, *rating*:integer, *age*:real)
Reserves(*sid*:integer, *bid*:integer, *day*:dates, *rname*:string)

- רשומת Reserves
  - באורך 40 בתים,
  - 100 רשומות בדף
  - ב-1000 דפים
- רשומת Sailors
  - באורך 50 בתים,
  - 80 רשומות בדף
  - ב-500 דפים

sid פירושו sailor id
bid פירושו boat id
sid הוא מפתח של Sailors
sid,bid,day הוא מפתח של Reserves
rname הוא שם המזמין (לאו דווקא השם של *sid*)

3

שיטות לביצוע פעולת הבחירה

4

פעולת בחירה עם תנאי אחד

- רוצים לחשב ביטויים מהצורה: $\sigma_{A\ op\ c}(R)$
  - כאשר A אטריביוט של R
  - op הוא אופרטור כגון >, =
  - c הוא קבוע
- לדוגמא, רוצים לחשב את השאילתה הבאה

```
SELECT *
  FROM Reserves R
 WHERE R.rname = 'Joe'
```

5

אין אינדקס, אין סדר

- ניתן לבצע את החישוב על ידי סריקה של היחס
- אם ביחס יש M דפים, נדרש מחיר של קריאת M דפים
- בדוגמא מהשקף הקודם, המחיר הוא קריאת 1000 דפים

6

אין אינדקס, יש סדר

- אם היחס R ממוין לפי אטריביוט A, אז ניתן
  - למצוא את הרשומה הראשונה שמקיימת את התנאי על ידי חיפוש בינארי
  - להמשיך כל עוד הרשומות מקיימות את התנאי
- המחיר:

log(M) + #pages with tuples from the result

7

שימוש באינדקס מסוג עץ +B לצורך ביצוע פעולת בחירה

8

שימוש באינדקס מבוסס עץ +B

- נניח שקיים אינדקס על עמודה A של R הממומש כעץ +B
- מוצאים כניסה ראשונה בעץ שתואמת את תנאי החיפוש
- ממשיכים לעבור באופן סדרתי על העלים של האינדקס כל עוד הערכים מקיימים את תנאי החיפוש
- מביאים את הרשומות המתאימות מהקובץ

9

דוגמה

- מבצעים בחירה עם התנאי *rname*<'C' על Reserves
- בהנחת התפלגות אחידה, מספר הרשומות ביחס Reserves שעונות על התנאי הוא:

  2/26 ≅ 10% ➔10,000 tuples
- נתוני הכניסה של 10,000 הרשומות הללו מאוחסנים ברצף במספר קטן של עלי האינדקס
- לכל נתון כניסה, קוראים את הדף המתאים של היחס

10

אם האינדקס מקובץ, אז

- נתוני כניסה סמוכים זה לזה מצביעים על רשומות שנמצאות באותו דף של היחס Reserves
- לכן, 10,000 הרשומות של היחס שעונות על התנאי תופסות 100 דפים (יש 100 רשומות בדף)
- בנוסף, כל אחד מ- 100 הדפים נקרא לתוך הזיכרון הפנימי (לחוצץ) פעם אחת בלבד
  - לפיכך אם האינדקס מקובץ, קוראים 100 דפים מהיחס Reserves

11

מדוע קוראים כל דף פעם אחת בלבד?

- נניח שקוראים לראשונה מהדיסק דף של היחס שיש בו רשומה שעונה על התנאי
- דף זה נקרא עבור נתון כניסה מסוים, שמופיע באחד העלים של האינדקס
- מכיוון שהאינדקס מקובץ, מיד לאחר נתון כניסה זה מופיעים ברצף נתוני הכניסה עבור כל שאר הרשומות שיש לקרוא מאותו הדף
- אבל הדף כבר בזיכרון הפנימי ואין צורך לקרוא אותו שוב

12

אם האינדקס אינו מקובץ, אז

- נתוני כניסה סמוכים זה לזה מצביעים על רשומות שנמצאות בדפים שונים של היחס Reserves
- לכן במקרה הגרוע ביותר, 10,000 הרשומות של היחס שעונות על התנאי נמצאות ב- 10,000 דפים שונים
- אבל ביחס Reserves יש 1,000 דפים
- לכן הרשומות שעונות על התנאי נמצאות ב- 1,000 דפים שונים לכל היותר

13

כמה דפים של היחס צריך לקרוא מהדיסק?

- כאמור, הרשומות של היחס Reserves שעונות על התנאי נמצאות ב- 1,000 דפים שונים לכל היותר
- אבל במקרה הגרוע ביותר, קוראים דף מהדיסק עבור כל רשומה שעונה על התנאי
  - לפיכך אם האינדקס אינו מקובץ, קוראים במקרה הגרוע ביותר 10,000 דפים של היחס Reserves

14

מדוע כל דף שמכיל רשומות של התוצאה נקרא הרבה פעמים?

- נניח שעבור נתון כניסה מסוים, דף של היחס נקרא לראשונה מהדיסק
- מכיוון שהאינדקס אינו מקובץ, מיד לאחר נתון כניסה זה מופיעים נתוני כניסה עבור רשומות שנמצאות על דפים אחרים
- לכן הדף שכבר נמצא בזיכרון מפנה את מקומו לדפים אחרים
- כשמגיעים שוב לנתון כניסה המתייחס לאותו הדף, צריך לקרוא את הדף שנית מהדיסק

15

אפשר לשפר

- תחילה קוראים את כל נתוני הכניסה שעונים על התנאי וממיינים אותם לפי מספר הדף
- בשלב השני קוראים את הדפים לפי הסדר הממוין
- מספר הדפים שיש לקרוא (במקרה הגרוע ביותר) הוא כמספר הרשומות שעונות על התנאי, אבל לא יותר ממספר הדפים ביחס
  - לפיכך, עם השיפור הזה, מספר הדפים שיש לקרוא במקרה הגרוע ביותר הוא 1,000
  - למרות השיפור, יתכן שעדיף לעבור על היחס באופן סדרתי (בלי שימוש באינדקס)

16

כמה דפים צריך לקרוא מהאינדקס?

- מכל רמה, חוץ מרמת העלים, קוראים לכל היותר דף אחד
  - יש סיכוי טוב שהדפים של הרמות העליונות כבר נמצאים בחוצץ (קרי, בזיכרון הפנימי)
- בכל מקרה, לרוב האינדקסים המבוססים על עץ +B יש מספר קטן של רמות
- הרשומות בעלים של אינדקס הן קצרות ויכולות להיות אלפי רשומות בדף אחד
- לפיכך, מספר הרשומות שקוראים מהאינדקס הוא בדר"כ בסדר גודל פחות ממספר הרשומות שיש לקרוא מהיחס עצמו (אין זה משנה אם האינדקס מקובץ או לא)

17

שימוש באינדקס ערבול לצורך ביצוע פעולת בחירה

18

שימוש באינדקס ערבול (Hash)

- אינדקס ערבול מתאים לצורך חישוב שאילתות בחירה כאשר התנאי הוא שוויון
- מציאת הדלי המתאים: כ- 2-1 גישות לדיסק
  - קריאה של השורש (directory) של מבנה הערבול (קריאה זו נחסכת אם השורש כבר בזיכרון הפנימי)
  - קריאה נוספת של הדלי המתאים
    - סביר להניח שאין יותר מדלי אחד שמתאים לשוויון (אחרת, עדיף שלא להשתמש בקובץ ערבול, כדי להימנע מ- overflow)
- קריאת הדפים של היחס לפי נתוני הכניסה שנמצאים בדלי ומקיימים את השוויון

דוגמה

- נניח שיש בחירה מהיחס Reserves לפי התנאי *rname*='Joe'
- נניח שיש אינדקס ערבול לא מקובץ על *rname*
- נניח שיש 100 הזמנות שנעשו על ידי Joe
- לפיכך, צריך לשלוף מהיחס Reserves 100 רשומות

המשך הדוגמה

- 100 הרשומות שצריך לשלוף מהיחס Reserves נמצאות
  - במקרה הגרוע ביותר ב- 100 דפים שונים
  - ובמקרה הטוב ביותר בדף אחד (כזכור, דף של היחס מכיל 100 רשומות)
- כלומר, בסכ"ה יהיו בין 2 ל-102 גישות לדיסק (כולל 2-1 גישות לקובץ העירבול)

שיטות לביצוע הטלה

שיטות לביצוע הטלה

- חישוב הטלה דורש מעבר על הקובץ
  - הורדת אטריביוטים לא רצויים
  - הורדת כפילויות
- הורדת כפילויות על ידי
  - שימוש במיון
  - שימוש בערבול (hashing)
- אפשר לבצע רק באמצעות אינדקס, במקרה של הטלה על עמודות שכולן נכללות במפתח החיפוש של האינדקס

שיטות לחישוב צירוף

דוגמה

- מעונינים לחשב את השאילתה הבאה

```
SELECT *
  FROM Reserves R, Sailors S
 WHERE R.sid = S.sid
```

25

שיטת חישוב נאיבית

- Simple Nested-Loops Join

```
foreach tuple r of of R do
     foreach tuple s of S do
          if r_i == s_j then
               add <r,s> to result
```

26

הערכת עלות

- נניח M בלוקים ב-R ו-N בלוקים ב-S וננית p_R רשומות לבלוק ב-R ו-p_S רשומות לבלוק ב-S
- עלות החישוב: $M + p_R*M*N$
  - לא כולל המחיר של כתיבת התוצאה לדיסק
- בדוגמה של השייטים:

$1{,}000 + 100*1{,}000*500 = 1{,}000 + 5*10^7$ I/Os

- בהנחה של 10ms זמן קריאת בלוק מהדיסק, יידרשו כ-140 שעות לביצוע החישוב!

27

שיטות עיקריות לחישוב הצירוף של שני יחסים

- צירוף היא הפעולה האלגברית היקרה ביותר וקיימות עבורה מספר שיטות חישוב שונות:
  - Block Nested-Loops Join
  - Index Nested-Loops Join
  - Sort-Merge Join
  - Hash Join
- להלן, נתאר את כל אחת מארבעת השיטות האלו

28

חישוב הצירוף כשיש מספיק מקום בזיכרון הפנימי לאחד משני היחסים

- רוצים לחשב את הצירוף הטבעי של R ו- S
- אם יש מספיק זיכרון, קרא את R בשלמותו לזיכרון
- לכל בלוק של S, קרא את הבלוק לזיכרון וחשב את הצירוף של הרשומות מבלוק זה עם כל הרשומות של R
- כתוב את התוצאה לדיסק וקרא את הבלוק הבא של S

29

כמות הזיכרון הפנימי הנדרשת

- מספר הבלוקים של הקטן מבין שני היחסים + 2
  - בלוק אחד לקריאת היחס השני
  - בלוק שני לכתיבת התוצאה
    - כשהוא מתמלא, כותבים אותו לדיסק ומתחילים למלאו מחדש

30

זמן חישוב כולל

- B_R בלוקים ב- R ו- B_S בלוקים ב- S
- כל בלוק של R ו- S נקרא פעם אחת
- כל בלוק של התוצאה נכתב פעם אחת
- זמן כולל: $B_R + B_S + output\ size$
- נתעלם מהזמן הדרוש לכתיבת התוצאה, כי הוא זהה בכל השיטות
- לכן הזמן הוא $B_R + B_S$

31

מה עושים אם כל אחד משני היחסים גדול מהזיכרון הפנימי?

- בזיכרון הפנימי יש B בלוקים
- קרא את R בחלקים – כל פעם B-2 בלוקים
- עבור כל חלק של R, קוראים את כל S, בלוק אחד בכל פעם
- חשב את הצירוף של הרשומות מהבלוק של S עם הרשומות מהחלק של R שנמצא בזיכרון
- יש $ceil(B_R/(B-2))$ איטרציות שבהן קוראים את כל S, בעוד ש- R נקרא פעם אחת
- הזמן הנדרש הוא $B_R + B_S * ceil(B_R/(B-2))$

32

הערה

- הזמן הוא $B_R + B_S * ceil(B_R/(B-2))$
- האם עדיף שבלולאה החיצונית יהיה היחס הקטן או היחס הגדול?
- תשובה: היחס הקטן (מדוע?)

33

למעשה תיארנו זה עתה את השיטה Block Nested-Loops Join

- Suppose that there are B buffer pages

```
foreach block of B-2 pages of R do
     foreach page of S do {
          for all matching in-memory
          pairs r, s:
               add <r,s> to result
     }
```

34

דוגמה להערכת העלות של Block Nested-Loops Join

- משתמשים ב-block nested-loops join לחישוב הצירוף מהדוגמה הקודמת תוך שימוש ב-102 מקומות לדפים בזיכרון
- בהנחה ש-Reserves ביחס החיצוני
  - נדרשות 1,000 גישות לקריאת Reserves
  - לכל 100 בלוקים של Reserves מעבר על כל היחס Sailors, לכן בסה"כ 10 מעברים על היחס Sailors ובכל מעבר קוראים 500 בלוקים
  - סך הכל 1,000 + 10*500 = 6,000 I/Os

35

המשך הערכת העלות

- כאשר Reserves בלולאה החיצונית נדרשו
  1,000 + (1,000/100)*500 = 6,000 I/Os
- כאשר Sailors בלולאה החיצונית יידרשו
  500 + (500/100)*1,000 = 5,500 I/Os
- בהנחה שקריאת בלוק אורכת 10ms, תידרש קצת יותר מדקה לחישוב הצירוף!

36

Index Nested-Loops Join

- נניח שיש אינדקס של S על אחד האטריביוטים שלפיהם מבצעים את הצירוף של S עם R

```
foreach tuple r of R
   foreach tuple s of S where r_i=s_j
      add <r,s> to result
```

- משתמשים באינדקס למציאת הרשומות המתאימות!

37

חישוב צירוף בעזרת אינדקס

- חישוב צירוף של R(A,B) עם S(B,C) כאשר יש אינדקס על עמודה B של S:
  - קרא בלוק של R ולכל רשומה מהבלוק
  - מצא בעזרת האינדקס את כל הרשומות המתאימות של S
  - הערה: קוראים את R פעם אחת בלבד

38

הערכת עלות

- בהינתן רשומה של R, נסמן ע"י X את הזמן הנדרש כדי למצוא את כל הרשומות המתאימות של S
- הזמן הכולל $B_R + t_R X$ (t_R מס' הרשומות ב- R)

39

מהו X?

- נניח שהאינדקס מכיל נתוני כניסה מהצורה (k, rid)
- גם אם יש הרבה נתוני כניסה עם אותו k עדיין סביר להניח שכולם על בלוק אחד, שיקרא b
- באינדקס ערבול צריך, בממוצע, לקרוא 1.2 בלוקים כדי להגיע לבלוק b
- באינדקס, שהוא עץ +B, צריך לקרוא 2-4 בלוקים כדי להגיע לבלוק b

40

המשך החישוב של X

- צריך לקרוא גם את הרשומות עצמן מהיחס S
- אם האינדקס מקובץ, אפשר להניח (שבדר"כ) כולן על בלוק אחד של S ולכן
  - צריך לקרוא בלוק אחד של S
- אם האינדקס אינו מקובץ, אז כל רשומה מתאימה של S נמצאת על בלוק נפרד, ומספר הבלוקים של S שצריך לקרוא שווה למספר הרשומות של S שמתאימות לרשומה אחת של R (נתאר בהמשך איך להעריך מספר זה)

41

לדוגמה

- אם מדובר באינדקס ערבול מקובץ, אז X=2.2 והזמן הכולל הוא

$$B_R + 2.2t_R$$

- לעומת $B_R + B_S * ceil(B_R / (B-2))$ בשיטה של Block Nested-Loops Join

42

דוגמה להערכת העלות של Index Nested-Loops Join

- נניח שליחס Sailors יש אינדקס ערבול על sid
- sid מפתח של Sailors ולכן לכל רשומת Reserves יש לפחות רשומה מתאימה אחת מהיחס Sailors (למעשה, בדיוק אחת – למה?)
- סריקת Reserves דורשת קריאת 1,000 בלוקים
- יש 100,000 = 100*1,000 רשומות ביחס Reserves
- לכל רשומה של Sailors, החיפוש באינדקס דורש לקרוא בממוצע 1.2 בלוקים

43

המשך הדוגמה

- סך כל העלות הנדרשת:
  1,000 + 100,000 * 2.2 = 221,000 I/Os
- סך כל הזמן הנדרש, בהנחה שקריאת בלוק אורכת 10ms, הוא כ- 35 דקות

44

דוגמה נוספת

- נניח שליחס Reserves יש אינדקס ערבול על sid
- סריקת Sailors דורשת קריאת 500 בלוקים
- יש 40,000 = 500*80 רשומות ב-Sailors
- יש 100,000 הזמנות ל-40,000 שייטים – בהנחה שההתפלגות אחידה: 2.5 הזמנות לשייט
- בהנחה שהאינדקס אינו מקובץ:
  500 + 40,000*(1.2 + 2.5) = 148,500 I/Os
  - החישוב ידרוש כ- 25 דקות

45

Sort-Merge Join

- חישוב צירוף של R(A,B) עם S(B,C) ע"י Sort-Merge Join:
  - מניחים כ"א מהיחסים על B ואז ממזגים:
    - עוברים על R עד ש- R.B >= S.B
    - עוברים על S עד ש- S.B >= R.B
    - חוזרים על שני הצעדים הקודמים עד ש- R.B = S.B, ואז קוראים לזיכרון את כל החלק של R וכל החלק של S ששווים על B ומחשבים את הצירוף בין שני חלקים אלה

46

sid	sname	rating	age
22	dustin	7	45
28	yuppy	9	35
31	lubber	8	55
36	lubber	6	36
44	guppy	5	35
58	rusty	10	35

Sailors

sid	bid	day	agent
28	103	12/4/96	Joe
28	103	11/3/96	Frank
31	101	10/2/96	Joe
31	102	12/7/96	Sam
31	101	13/7/96	Sam
58	103	22/6/96	Frank

Reserves

47

זמן החישוב של צירוף ע"י Sort-Merge Join

- זמן למיון:
  $B_R \log B_R + B_S \log B_S$
- זמן למיזוג (תחת איזה הנחה?)
  $B_R + B_S$
- סה"כ:
  $B_R \log B_R + B_S \log B_S + B_R + B_S$

48

דוגמה

- מיון Reserves דורש
  1000 log 1000 ≅ 10,000 I/Os
- מיון Sailors דורש
  500 log 500 ≅ 4,500 I/Os
- צירוף: 1,000 + 500 = 1,500 I/Os
- סה"כ: 10,000 + 4,500 + 1,500 = 16,000 I/Os
- הזמן שנדרש יהיה פחות משלוש דקות
- על ידי שימוש בשיטות מיון טובות, ניתן להקטין את עלות החישוב לפחות מחצי העלות המחושבת כאן

Hash Join

- מבצעים חלוקה של שני יחסי הקלט ל- B-1 חלקים ע"י הפעלת פונקצית ערבול על ערכי האטריביוטים שלפיהם נעשה הצירוף
  - על פי החלוקה, רשומות של R שמופו לקבוצה i יכולות להתחבר רק לרשומות של S שאף הן מופו ל-i
- קוראים את קבוצות החלוקה של R בזו אחר זו וכל קבוצה מחלקים לתת קבוצות (שנשמרות בזיכרון הראשי) בעזרת פונקצית ערבול חדשה (שונה מהראשונה!)
- לכל קבוצת חלוקה של S קוראים את הקבוצות המתאימות של R ומנסים למצוא התאמות

Hash-Join Algorithm

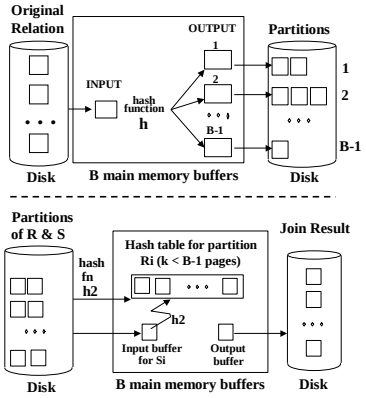
```
//Partition R into k partitions
foreach tuple r in R do
  read r and add it to buffer page h(r_i)
       //flush buffer page when it fills

//Partition S into k partitions
foreach tuple s in S do
  read s and add it to buffer page h(s_j)
       //flush buffer page when it fills
```


Hash-Join Algorithm (cont'd)

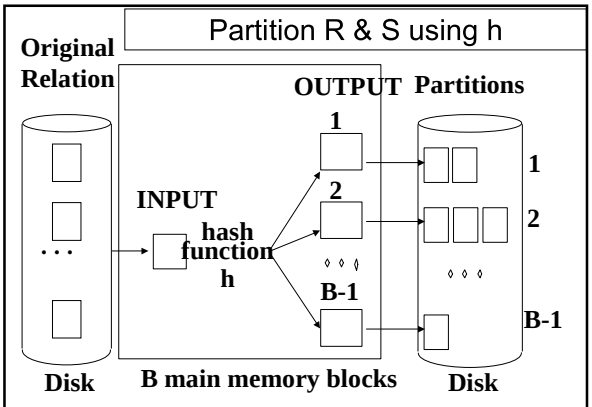
```
//Probing phase

for l = 1..k
  //Build in-memory hash table for R_l
  //using h2(r_i)
  foreach tuple r in partition R_l do
    read r and insert into hash table
    using h2(r_i)
 //Scan S_l and probe for matching R_l tuples
  foreach tuple s in partition S_l do
    read s and probe table using h2(s_j)
    output matching pairs <r,s>
```

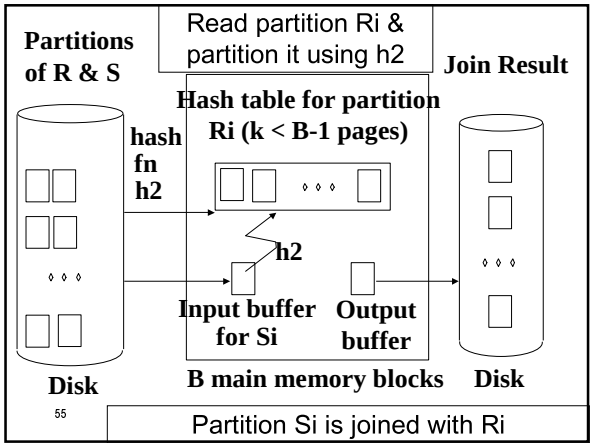

Hash Join

- Partition both relations using hash fn **h**: R tuples in partition i will only match S tuples in partition i
- Read in a partition of R, hash it using **h2 (<> h!)**. Scan matching partition of S, search for matches



Partition R & S using h





55

המחיר של Hash Join

- אם יש מספיק מקום בזיכרון וה-Partitions הן פחות או יותר בגודל שווה, אז
  - בשלב הראשון קוראים וכותבים כל יחס פעם אחת בלבד
  - ובשלב השני קוראים כל יחס פעם אחת
- לכן המחיר הכולל הוא $3(B_R + B_S)$

56

בדוגמת השייטים

- עבור 1,000 בלוקים של Reserves ו-500 בלוקים של Sailors יידרשו
  $3(1{,}000 + 500) = 4{,}500$ I/Os
- במקרה זה, לביצוע הצירוף יידרשו כ-45 שניות

57

כמות הזיכרון הנדרשת לביצוע Hash Join

- אם יש B בלוקים בזיכרון אז מספר ה-Partitions הוא לכל היותר k=B-1
- לכן הגודל של כל Partition הוא $\frac{B_R}{B-1}$
- לכן צריך בשלב השני $\frac{fB_R}{B-1}$ בלוקים לכל Partition, כאשר f הוא מקדם גדול מ- 1, כי בשלב השני צריך לייצר מבנה hash לפי h2 וזה דורש קצת יותר מ- B-1 בלוקים

58

המשך חישוב כמות הזיכרון הנדרשת לביצוע Hash Join

- כאמור, בשלב השני צריך $\frac{fB_R}{B-1}$ בלוקים
- יש בסך הכל B בלוקים וצריך גם אחד לכתיבת התוצאה ואחד לקריאת היחס השני, ולכן
$$B > \frac{fB_R}{B-1} + 2$$
- לפיכך, צריך $B > \sqrt{fB_R}$ בלוקים, כאשר R היחס הקטן יותר

למעשה צריך קצת יותר, כי ה- Partitions לא בגודל שווה

59

שיטה משופרת ל-Sort-Merge Join (SMJ)

- יש שיטה משופרת ל- SMJ שלוקחת אותו זמן כמו hash join: $3(B_R + B_S)$
- שיטה זאת דורשת $B > \sqrt{B_s}$ בלוקים בזיכרון, כאשר S הוא הגדול משני היחסים (פרטים בפרק 12)
- SMJ מייצרת תוצאה ממוינת וזמן החישוב הוא worst case

60

סיכום

- ראינו שיטות שונות לחישוב צירוף של שני יחסים
- לכל שיטה נוסחה עבור זמן החישוב
  - אין שיטה שהיא תמיד הכי טובה

61