

מבוא לתכנות

הרצאה 12 – אלגוריתמים וסיבוכיות
סמסטר 1

- להבין חיפוש מהו ולהבין את האלגוריתמים לחיפוש ליניארי ובינארי.
- ללמוד 2 דרכים למיון: מיון בחירה ומיון מיזוג.
- להבין את הטכניקות הבסיסיות לניתוח יעילות אלגוריתמים באמצעות שימוש באלגוריתמי חיפוש ומיון.

מהלך ההרצאה

- אלגוריתמים של חיפוש ליניארי ובינארי.
- סיבוכיות אלגוריתמי החיפוש.
- אלגוריתמי מיון בחירה ומיזוג.
- סיבוכיות אלגוריתמי מיון.

- הבעיה: בהינתן רשימת מספרים בסדר אקראי, נתבקשנו לבדוק האם קיים בה מספר מסוים כלשהו (נגיד 6)?
- כיצד אתם הייתם עושים זאת?

חיפוש ליניארי

- חיפוש ברשימה איבר איבר עד שהמספר המבוקש נמצא.

```
static int LinearSearch(int x, int[] nums)
{
    for (int i = 0; i < nums.Length; i++)
        if (nums[i] == x) // item found, return the index value
            return i;
    return -1;           // loop finished, item was not in list
}
```

חיפוש ליניארי

- אם הרשימה ממוינת בסדר עולה (מהנמוך לגבוה), נוכל לחסוך חלק מתהליך החיפוש.
- ברגע שמצאנו ערך שגדול מהמספר המבוקש, ניתן לעצור את החיפוש הליניארי מבלי לבדוק את שאר הרשימה.
- בממוצע, זה יחסוך לנו כחצי מהעבודה.

חיפוש בינארי

- עובד רק כאשר הנתונים ממוינים!
- בחיפוש בינארי, מתחילים מאמצע הרשימה. אם מצאנו את המספר, סיימנו. אחרת, נמשיך לחפש רק בחלק אחד מבין השניים (המספרים הגדולים או הקטנים)
- בינארי משמעותו "שתיים", ובכל צעד אנו מחלקים את קבוצת המספרים לבדיקה ל-2 חלקים.

חיפוש בינארי

• אלגוריתם חיפוש בינארי

```
static int BinarySearch(int x, int[] nums)
{
    int low=0 , mid, item;
    int high = nums.Length - 1;
    while (low <= high)           // There is still a range to search
    {
        mid = (low + high) / 2; // Position of middle item
        item = nums[mid];
        if (x == item)           // Found it! Return the index
            return mid;
        else if (x < item)       // x is in lower half of range
            high = mid - 1;      // move top marker down
        else                     // x is in upper half of range
            low = mid + 1;       // move bottom marker up
    }
    return -1;                   // No range left to search,
                                // x is not there
}
```


מהלך ההרצאה

- אלגוריתמים של חיפוש ליניארי ובינארי.
- סיבוכיות אלגוריתמי החיפוש.
- אלגוריתמי מיון בחירה ומיזוג.
- סיבוכיות אלגוריתמי מיון.

סיבוכיות

- איזה אלגוריתם חיפוש טוב יותר, ליניארי או בינארי?
 - החיפוש הליניארי קל יותר להבנה ולמימוש.
 - החיפוש הבינארי יעיל יותר מכיוון שהוא לא צריך לחפש בכל הרשימה כולה.
- כדי לחשב מי טוב יותר נצטרך לחשב את סיבוכיות הזמן (והמקום) של כל אחד. האחד עם הסיבוכיות הנמוכה יותר – טוב יותר.
- הכוונה בסיבוכיות היא מספר ה"צעדים" הנדרשים לסיום המטלה. פחות צעדים = אלגוריתם יעיל יותר.


סיבוכיות

- הסיבוכיות מחושבת ביחס לגודל הקלט.
- אם גודל הקלט הוא n , כמה צעדים נצטרך?
- מה בנוגע ל- n גדול יותר? $(100, 1000, \dots, 100000000)$?
- שימו לב שיש לחשב סיבוכיות עבור "המקרה הגרוע ביותר" ולא המקרה הטוב ביותר, הלא הוא 1, עבור החיפוש.

- בואו נבחן את החיפוש הליניארי.
 - עבור רשימה של 10 מספרים נצטרך לעבור 10 פעמים.
 - עבור רשימה בגודל כפול, נצטרך לעבור 20 פעמים.
 - עבור רשימה בגודל פי 3, נצטרך לעבור 30 פעמים!
- כמות הזמן הנדרש קשורה ליניארית לגודל הקלט n.
זה נקרא: אלגוריתם זמן ליניארי.

- כעת נבחן חיפוש בינארי.
 - נניח וישנם 16 מספרים. כל סיבוב בלולאה מוריד את זה בחצי. לאחר סיבוב אחד, יהיו 8 מספרים.
 - אחרי 2 סיבובים, 4 מספרים נותרו.
 - אחרי 3 סיבובים, 2 מספרים נותרו.
 - אחרי 4 סיבובים, רק מספר 1 נותר.
- אם חיפוש בינארי נמשך i סיבובים, הוא מסוגל למצוא ערך אחד ברשימה בגודל 2^i .

סיבוכיות

- כדי לקבוע כמה איברים נבחנו מרשימה בגודל n , נצטרך לפתור עבור i , או $n = 2^i$ $i = \log_2 n$.
- חיפוש בינארי הוא דוגמה לאלגוריתם בזמן \log .
- $\text{Log}(1,048,576) = 20$

לעומת המקרה הממוצע בליניארי 500,000
- אז הוא חזק מאוד!!!

מהלך ההרצאה

- אלגוריתמים של חיפוש ליניארי ובינארי.
- סיבוכיות אלגוריתמי החיפוש.
- **אלגוריתמי מיון בחירה ומיזוג.**
- סיבוכיות אלגוריתמי מיון.

- הבעיה הבסיסית במיון היא לקחת רשימת מספרים ולסדר אותם מחדש בסדר עולה (או יורד).
- כיצד אתם הייתם עושים זאת?

מיון בחירה

- האלגוריתם רץ בלולאה, ובכל סיבוב בלולאה האיבר הקטן ביותר נבחר ומועבר למקומו המיועד.
 - עבור n אלמנטים, נמצא את הערך הקטן ביותר ונשים אותו במקום 0.
 - לאחר מכן, נתחיל לחפש את האיבר הקטן ביותר החל ממקום 1 ועד $(n-1)$ ונשים אותו במקום 1.
 - וכך הלאה...

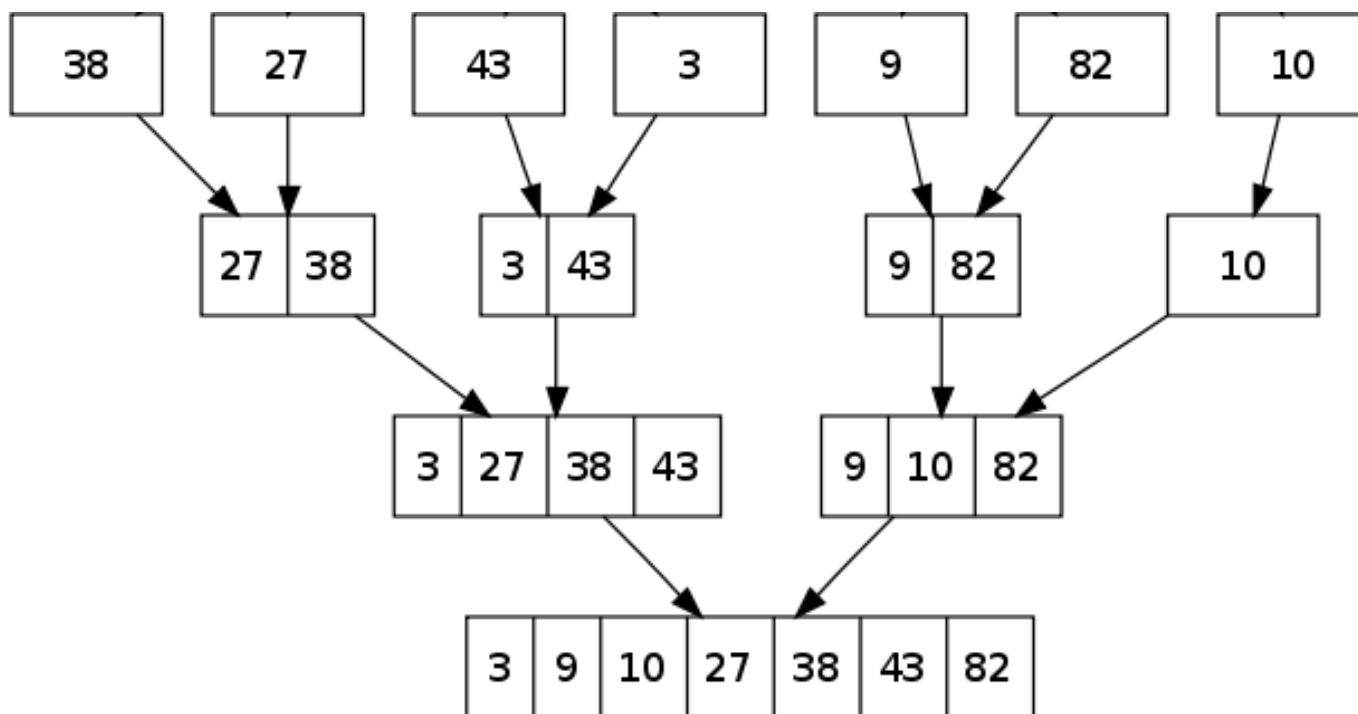
• אלגוריתם מיון בחירה

```
static void SelectionSort(int[] nums)
{
    int n = nums.Length;
    for (int bottom = 0; bottom < n - 1; bottom++)
    {
        for (int i = bottom + 1; i < n; i++)
        {
            if (nums[i] < nums[bottom])
            {
                int temp = nums[bottom];
                nums[bottom] = nums[i];
                nums[i] = temp;
            }
        }
    }
}
```

מיון מיזוג

- מיון הבחירה היה פשוט למימוש, אך לא יעיל במיוחד. כעת נלמד את אלגוריתם מיון מיזוג הטוב יותר, אך הקשה יותר למימוש. הרעיון המרכזי כאן יהיה אסטרטגית "הפרד ומשול".
- זה לוקח מספר מעברים על הנתונים. בכל מעבר, המערך מחולק לבלוקים בגודל m . בהתחלה $m=1$. כל 2 בלוקים צמודים ממוזגים יחדיו ואז המעבר הבא עם ערך m אשר גדול פי 2.

מיון מיזוג



מיון מיזוג

- נצטרך 2 פונקציות כאן. הראשונה כדי לפצל את הרשימה לבלוקים בגודל m .
 - MergeSort(array)
- השנייה תמזג בכל פעם 2 בלוקים צמודים.
 - MergeArrays(array, startL, stopL, startR, stopR)

- MergeSort

```
//Bottom-up merge sort
static void MergeSort(int[] array)
{
    if (array.Length < 2) //We consider the array already sorted, no change
        return;
    //The size of the sub-arrays . Constantly changing
    int step = 1;
    //startL - start index for left sub-array
    //startR - start index for the right sub-array
    while (step < array.Length)
    {
        int startL = 0;
        int startR = step;
        while (startR + step <= array.Length)
        {
            MergeArrays(array, startL, startL + step, startR, startR + step);
            startL = startR + step;
            startR = startL + step;
        }

        if (startR < array.Length)
        {
            MergeArrays(array, startL, startL + step, startR, array.Length);
        }
        Console.WriteLine("- - - with step = " + step);
        step *= 2;
    }
}
```

- MergeArrays

```
//Merge to already sorted blocks
static void MergeArrays(int[] array, int startL, int
stopL, int startR, int stopR)
{
    //Additional arrays needed for merging
    int[] right = new int[stopR - startR];
    int[] left = new int[stopL - startL];
    //Copy the elements to the additional arrays
    for (int a = 0; a < right.Length; a++)
        right[a] = array[startR + a];
    for (int b = 0; b < left.Length; b++)
        left[b] = array[startL + b];
}
```

• המשך בשקופית הבאה...

מיון מיזוג

```
int i = 0;
int j = 0;
int k = startL;
while (i < left.Length && j < right.Length)
{
    if (left[i] < right[j])
    {
        array[k] = left[i];
        k = k + 1;
        i = i + 1;
    }
    else
    {
        array[k] = right[j];
        k = k + 1;
        j = j + 1;
    }
}
while (i < left.Length)
{
    array[k] = left[i];
    k = k + 1;
    i = i + 1;
}
while (j < right.Length)
{
    array[k] = right[j];
    k = k + 1;
    j = j + 1;
}
ShowArray(array);
```


מהלך ההרצאה

- אלגוריתמים של חיפוש ליניארי ובינארי.
- סיבוכיות אלגוריתמי החיפוש.
- אלגוריתמי מיון בחירה ומיזוג.
- סיבוכיות אלגוריתמי מיון.

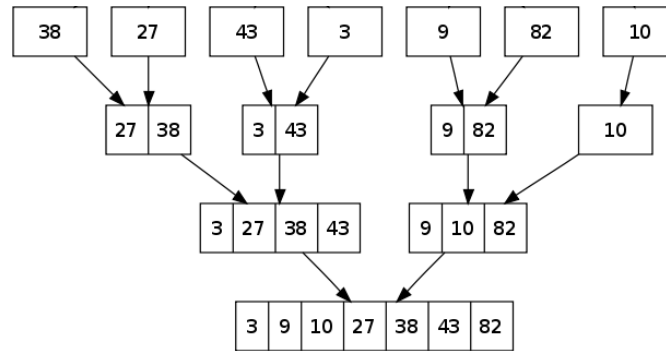
סיבוכיות

- בואו נתחיל עם מיון הבחירה
- נניח והתחלנו עם רשימה בגודל n . כדי למצוא את האיבר הקטן ביותר, האלגוריתם בודק את כל האיברים. בסיבוב הבא בלולאה, זה בודק את שאר ה- $n-1$ איברים. מספר החזרות יהיה:

$$n + (n-1) + (n-2) + (n-3) + \dots + 1 = \sum_{n=1}^N n = \frac{n(n+1)}{2} \approx n^2$$

- הנוסחה מכילה n^2 , כלומר מספר הצעדים באלגוריתם פרופורציונלי לריבוע גודל הרשימה. זה נקרא אלגוריתם בסיבוכיות n^2 .

סיבוכיות

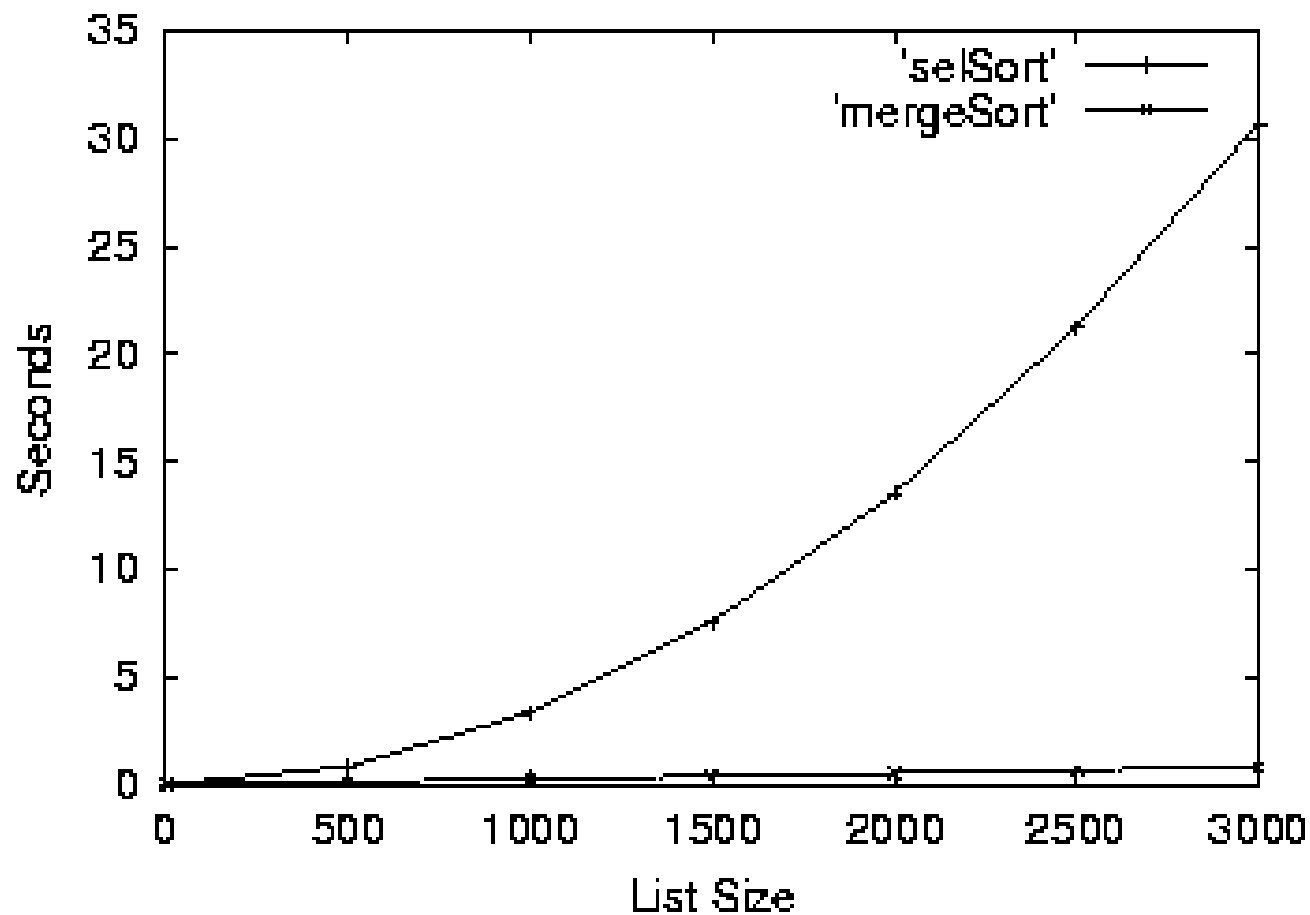


- כעת בואו נראה את מיון המיזוג.
- כל רמה של מיזוג כוללת העתקה של n ערכים. השאלה היא, כמה רמות ישנן?
- $\log_2 n$ – ולכן, כלל העבודה הדרושה למיון n איברים היא $n \log_2 n$. זה נקרא אלגוריתם בסיבוכיות $n \log(n)$.

סיבוכיות

- אז, מי טוב יותר, מיון הבחירה בסיבוכיות n^2 , או מיון המיזוג בסיבוכיות $n \log n$?
- אם הקלט קטן, מיון הבחירה עלול להיות הבחירה המתאימה כי הוא קל למימוש וצורך פחות משאבים.
- מה קורה כאשר n גדל? ראינו בדיון על החיפוש הבינארי שפונקציית ה- \log גדלה בקצב איטי מאוד, אז $n \log n$ יגדל בקצב איטי יותר מאשר n^2 .

סיבוכיות



- מדעי המחשב הם יותר מרק תכנות!
- המחשב החשוב ביותר עבור כל מתכנת מקצועי הוא האחד שנמצא בין האוזניים שלו.