Lecture 16

Lecture contents:

- Sort algorithms
  - Selection sort
  - Bubble sort
- Algorithm Complexity

احنا المحاضرة اللي فاتت كنا بنتكلم عن ال search algorithms زي ال linear & binary & hashing table
المرة دي بقى هنتكلم عن ال sort algorithms .. و له انواع مختلفة زي ال Selection sort & bubble sort
وفيه Insertion sort برضه والدكتور قال هيجيبه في الامتحان وناخد بالنا منه <3 .. بس هو مشرحش عنه حاجة في المحاضرة :v

المهم نبدأ بقى .. المحاضرة دي هنجيب فيها حاجات من على النت عشان توضح لنا الدنيا اكتر .. وهتحتاج تفتح ال slides برضه معلش :D
ودة اللينك بتاعها :
https://drive.google.com/open?id=1pAvhfQbnhjfhvfvghScVqh8_NIxTWdOU

Selection sort:

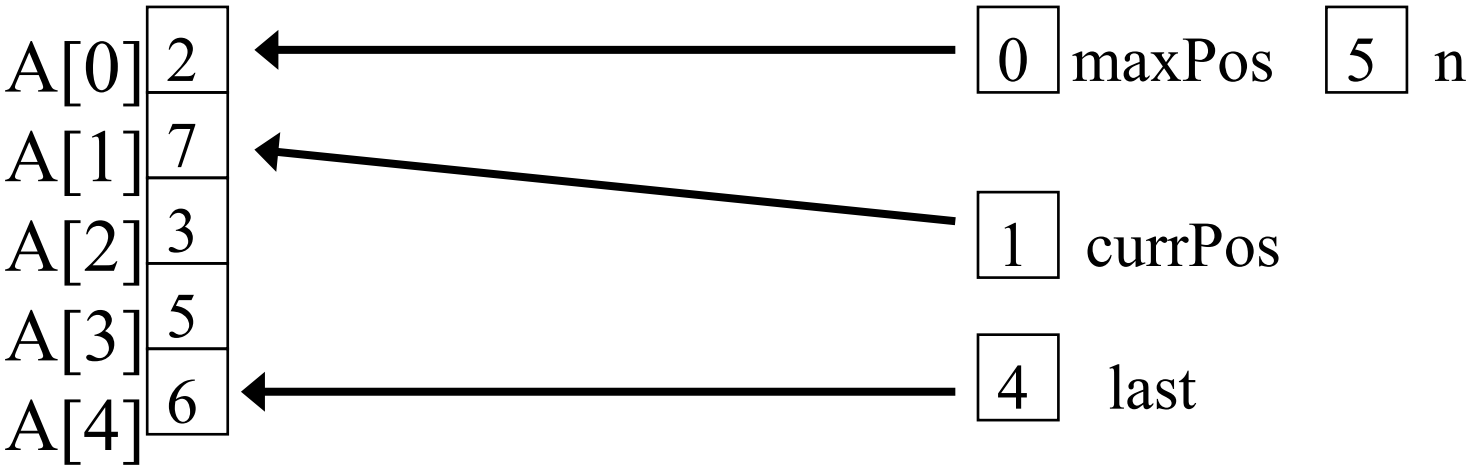
تخيل احنا معانا list فيها n elements وعايزين نرتب اللي فيها من الصغير للكبير ... فبـ 3 جنيه تفكير هنقول خلاص احنا هندور في ال list دي على اكبر element ولما نلاقيه ناخده ونحطه في اخر ال list .. وبعدها ادور تاني في ال list على تاني اكبر element ولما الاقيه احطه فوق اكبر element وهكذا لحد لما ال list كلها تترتب
الكلام اللي فوق دة يا معلم هو ال Selection sort algorithm
ودة بقى يودينا لسؤال عميق شوية .. هو يعني ايه algorithm اصلا؟؟
ال algorithm هو مجموعة خطوات **محددة** تؤدي لنتيجة معينة
يعني مينفعش اقول ان ال algorithm هو اني اقارن ال elements ببعضها وأ shift لحد ما ال list بتاعتي تترتب .. فمش هي دي مجموعة الخطوات **المحددة** اللي بنتكلم عنها .. لأن ببساطةدي الفكرة العامة من ال sorting اللي احنا بنطبقها في ال selection sort وفي ال bubble وال insertion وفي اي نوع من انواع ال sort اللي موجودة ... بس طريقة عمل نوع ال sort دة هي اللي بتختلف، فالطريقة ال unique دي هي ال algorithm
اشطة كدة؟؟ .. اشطة

ناخد مثال بقى على اللي عايزين نعمله .. موجود في slide 7

عندنا array من 5 elements وعايزين نرتبها ب selection sort :

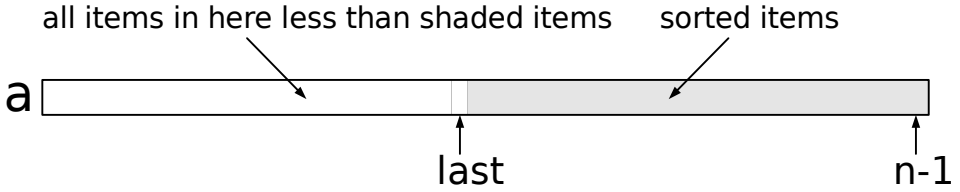
A[0]	2
A[1]	7
A[2]	3
A[3]	5
A[4]	6

- هنعمل بقى variable نخزن فيه عدد ال array elements اللي عندنا (اللي هو هنا 5)
- وزي ما قولنا محتاجين نمشي على كل element في ال array ونشوف مين اكبر element .. فمحتاجين variable يشيل ال element دة وهنسميه maxPos
- ومحتاجين counter عشان نعرف نمشي على ال array elements وهنسميه CurrPos
- طب تخيل دلوقتي احنا مشينا بال CurrPos على كل ال elements ولقينا ان اكبر element هو رقم 7 فهناخده ونحطه في اخر ال list والمفروض بعد كدة هنرجع تاني نخلي CurrPos يلف على ال array من اول وجديد .. بس احنا مش عايزين نقعد نمشي كل مرة على كل ال array elements ونقارنهم ببعض .. لأن ببساطة انا دلوقتي عارف ان ال 7 دي هي اكبر element خلاص فمش محتاج اقارنها تاني واضيع وقت على الفاضي، ودي هتبان اكتر لو طلعنا ال 6 وال 5 كمان فوق ال 7 .. فانا دلوقتي المفروض اني اقارن ال 2 بال 3 وبس (عشان خلاص قارنت الباقي) فمفيش اي داعي يخليني اقعد اقارن الحاجات اللي انا رتبتها خلاص ... فهنعمل variable كمان يقول لنا ان هو بيشاور على اول element مترتب خلاص .. وهنسميه last
وهيبقى دة شكل ال variables اللي عملناها وهي بتشاور على ال array :



ففي الاول زي ماهو واضح هنبدأ ال maxPos ب 0 و CurrPos ب 1 ونفضل نقارن ال maxPos بال CurrPos ببعض ونخلي ال CurrPos يشاور على ال element اللي بعديه لحد ما نوصل لل element اللي عندها ال currPos = last ونقارن برضه وبعد كدة نبقى خلاص عرفنا مين اكبر element فنقوم حاطينه في اخر ال array **اللي مش مترتبة** .. يعني نحطه في ال element اللي ال last بيشاور عليه (عن طريق اننا نبدل ال 2 elements ببعض .. هنعمل swap من الاخر)

معنى كدة ان احنا لو جينا نبص على ال list واحنا شغالين في ال sorting هنلاقيها اتقسمت لجزء مش مترتب وعايزين نرتبه وجزء مترتب فمش بنقربله تاني خلاص :



كدة احنا جيبنا اكبر element في ال array وحطيناه في المكان بتاعه (لو كان دة المطلوب فاحنا كدة نبقى خلصنا)
ودة الكود اللي بيجيب max element :

The maxSelect function

```
int maxSelect(int a[], int n)
{
  int maxPos(0), currentPos(1); //it's equivalent to : int maxPos = 0, currentPos=1;
  while (currentPos < n) {
  // Invariant: a[maxPos] >= a[0] ... a[currentPos-1]
    if (a[currentPos] > a[maxPos])
      maxPos = currentPos;
    currentPos++;
  }
  return maxPos;
}
```

بس لو المطلوب اننا نرتب ال array كلها فاحنا هنكرر نفس ال iteration دي وكل مرة هنقلل ال last ب 1 لحد ما نوصل ل last = 0 ..
يعني من الاخر كدة احنا بنعمل ال iteration دي (n-1) مرة عشان نرتب اي list بال selection sort
بص بقى على ال slides من اول slide 7 لحد 32 فيها step by step sorting لل example اللي فوق

ودة الكود

Selection Sort

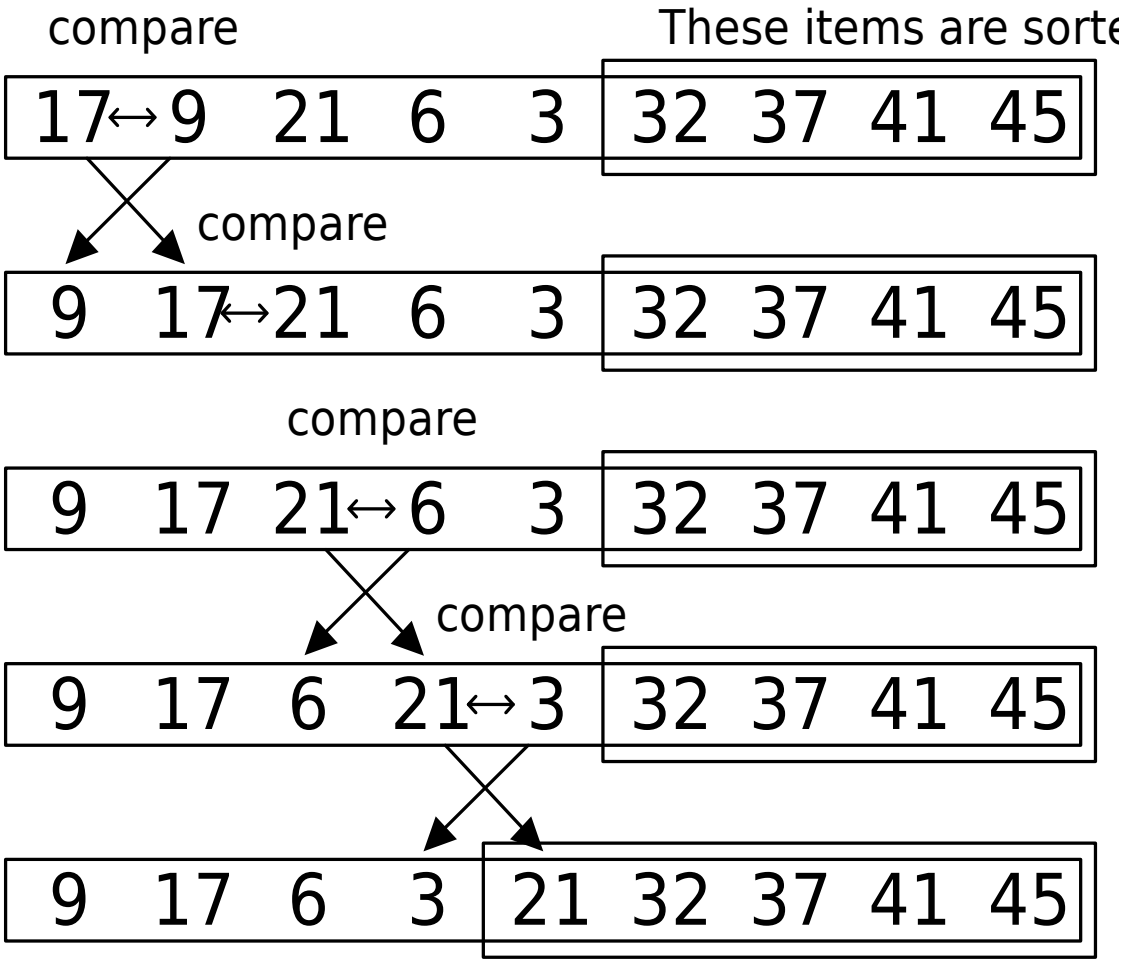
```
void selectionSort(int a[], int n)
{
  int last(n-1);
  int maxPos;
  while (last > 0) {
  // invariant: a[last+1] ... a[n-1] is sorted &&
  //    everything in a[0] ... a[last] <= everything in a[last+1] ... a[n-1]
    maxPos = maxSelect(a, last+1); // last+1 is length from 0 to last
    swapElements(a, maxPos, last);
    last--;
  }
}
```

دة كدة ال selection algorithm .. ممكن نلخصه في جملة واحدة ==> بـ select اكبر element عندي واحطه في اخر ال list
وطبعا اكيد احنا كان ممكن نقارن وندور على ال minPos بدل ال maxPos وساعتها هنعمل variable اسمه first بدل ما كان last
وبدل ما كنا بنبدأ المقارنة في ال array من اول element لحد ال last فاحنا دلوقتي هنبدأ من عند المكان اللي first بيشاور عليه وهنخلص
عند اخر ال array خالص .. بص على ال GIF دي بتوضح الدنيا اكتر لو فيه حاجة لسة واقعة:
https://en.wikipedia.org/wiki/File:Selection-Sort-Animation.gif
والبرنس اللي عمل اللينك دة عظيم ياخواننا والله :
http://www.cs.armstrong.edu/liang/animation/web/SelectionSort.html
ودة كمان:
https://www.toptal.com/developers/sorting-algorithms/selection-sort

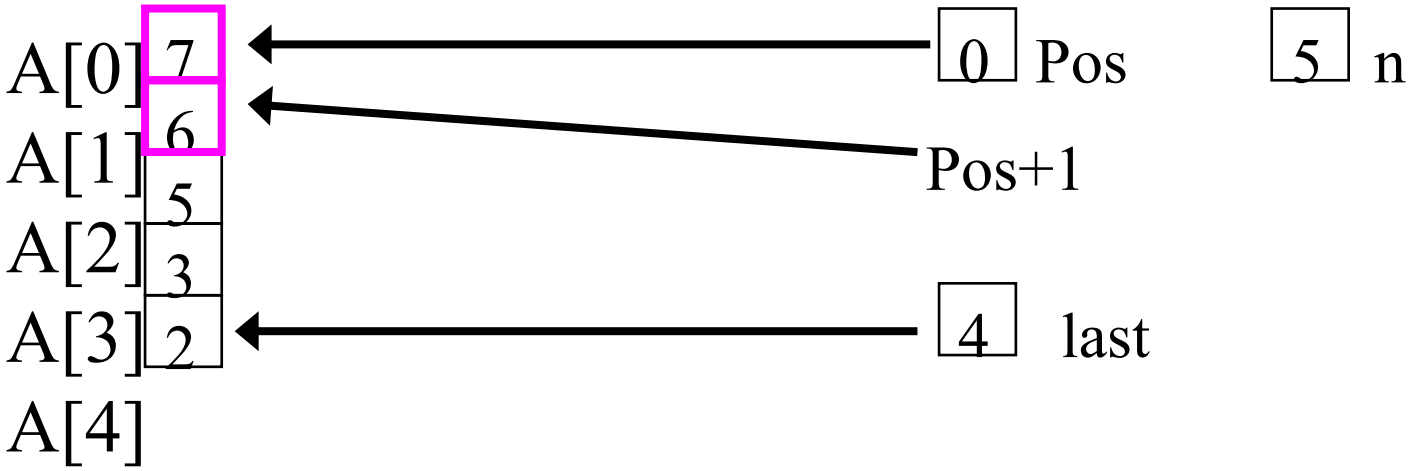
---

Bubble sort:

احنا في ال selection كنا بنعمل comparison على كل ال elements في ال array وبعد كدة بنـ shift .. هنا لأ، هنا احنا هنعمل
bubbles .. يعني كأننا بنحط كل 2 elements ورا بعض في 2 bubbles ونقارن .. لو ال bubble الاولانية اكبر من التانية فاحنا هنـ
swap ال 2 bubbles دول، ولو لأ هيبقوا زي ما هم .. وبعدين نقارن ال element التاني بالتالت ونشوف هنـ swap ولا لأ وبعد كدة
التالت بالرابع وهكذا لحد ما نوصل لل last .. وبكدة احنا عملنا اول iteration ، فهنقلل ال last بواحد ونرجع نقارن تاني وهكذا لحد ما
نوصل ل last = 0
بص على الصورة دي في slide 38 بتوضح ال swap اللي بيحصل :



تعالى بقى نفكر في ال implementation .. هنا احنا مش هنعمل maxPos بقى عشان مش هنستنى لآخر ال array وهنـ swap على طول لو محتاجين نعمل swap .. فهنعمل last & pos & n بس :



وبرضه هنا احنا بنعمل (n-1) iterations زي ال selection عشان نرتب ال list كلها .. بس احنا هنا بنعمل swapping كتير .. (ممكن نعمل swap في كل مرة بنعدي على ال elements في ال iteration الواحدة) .. فهو دة الفرق اللي بين ال selection & bubble

بص في ال slides من اول سلايد 43 لحد 57 عشان تشوف الترتيب بيتعمل ازاي .. وخلي بالك فيه غلطة في ال slides عشان ال last مش بيقل بعد كل iteration .. فهو المفروض هيقل 1 عند سلايد 47 وعند 52 وعند 55 و 56

وهذا هو الكود : (خلي بالك ال condition بتاع ال for loop فيه غلطة في السلايدز بس هنا متصلحة ((الكود اللي تحت دة صح)))

bubbleSortPhase

```
// void swapElements(int a[], int maxPos, int last);

void bubbleSortPhase(int a[], int last)
{
  // Precondition: a is an array indexed from a[0] to a[last]
  // Move the largest element between a[0] and a[last] into a[last],
  // by swapping out of order pairs
  int pos;

for (pos = 0; pos <= last - 1; pos++)
    if (a[pos] > a[pos+1]) {
      swapElements(a, pos, pos+1);
    }
  // Postconditions: a[0] ... a[last] // contain the same elements,
  // possibly reordered; a[last] >= a[0] ... a[last-1]
}
```

Bubble Sort

```
void bubbleSortPhase(int a[], int last);
void bubbleSort(int a[], int n)
{ // Precondition: a is an array indexed from a[0] to
a[n-1]
      int i;
      for (i = n - 1; i > 0; i--)
            bubbleSortPhase(a, i);
      // Postcondition: a is sorted
}
```

كدة خلصنا ال bubble sort وبرضه بص على ال GIF دة عشان الدنيا تثبت :
https://commons.wikimedia.org/wiki/File:Bubble-sort.gif
ودة اللينك العظيم بتاع البرنس :
http://www.cs.armstrong.edu/liang/animation/web/BubbleSort.html
ودة كمان:
https://www.toptal.com/developers/sorting-algorithms/bubble-sort

---

تعالى بقى نسرح شوية ونقول شوية خواطر نسلي بيهم صيامنا

خواطر

خاطرة رقم 1 ** :في ال **bubble sort : افرض مثلا جينا نقارن ال elements ببعضها ووصلنا لآخر ال array من غير ما يحصل ولا swap في ال iteration … يبقى ساعتها نقدر نستنتج ان ال array خلاص مترتبة فخلا مفيش اي داعي يخلينا نعمل كل ال iterations لحد ما نوصل ل last = 0 .. فاحنا ممكن نعمل early exit condition ونخرج من ال sorting operation بدري وبالتالي نوفر وقت و resources

**خاطرة رقم 2 ** : احنا عندنا algorithms كتير في ال sorting .. اخدنا منهم اتنين وفيه كتير مخدناهمش، فليه فيه كذا algorithm بيعملوا نفس الحاجة؟؟
الاجابة ان كل algorithm من اللي موجودين بيكون most efficient مع شكل معين من الداتا .. يعني بمعنى أصح ك programmer ببص على شكل الداتا اللي عايز اشتغل عليها وعلى اساس الشكل دة هحدد هشتغل باي algorithm .. فمثلاً لو انا عارف ان الداتا بتاعتي مش مترتبة خالص يبقى هختار ال algorithm X لأنه سريع مثلاً .. ولو الداتا اللي معايا عايز ارتبها تصاعدي هستخدم algorithm Y ولو الداتا فيها اصفار كتير هستخدم algorithm Z وهكذا

الكلام اللي جي دة الدكتور بيحب يجيبه في الامتحانات ... هنتكلم في حاجة اسمها algorithm complexity :

Algorithm complexity:

انا هبعد خالص هنا عن شرح المحاضرة وهحاول اشرح باسلوب مختلف من النت واربطه بأمثلة المحاضرة .. فلو حد شايف حاجة مفهومة او مشروحة غلط يقول

في ويكيبيديا بيقول ان ال complexity دي زي function بتوصف الوقت اللي بيتاخد او عدد ال operations اللي بتحصل عشان ال algorithm يتنفذ .. طب انا ليه بقول complexity وبتاع وماحسبش الوقت اللي الكود بياخده عشان يتنفذ وخلاص؟؟
هقولك لأ مش صح لأن ال data size بيتغير .. انا ممكن اشتغل على 10 elements بنفس ال algorithm اللي بشتغل بيه على داتا فيها مليون element وبالتالي الوقت هنا بيفرق كتير جدا عن الوقت في الحالة الاولانية .. وحتى لو الداتا ليها نفس ال size، يعني مثلا انا بعمل نفس ال sort algorithm على 2 lists ليهم نفس ال size، بس واحدة مترتبة والتانية مش مترتبة ... الاولانية مش هيحصل فيها اي swap وبالتالي هتخلص بسرعة، بس التانية هيحصل فيها swap عشان مش مترتبة وبالتالي هتاخد وقت اكبر من ال list الاولانية فماينفعش اوصف ال algorithm بالوقت بتاعه لأن صعب جدا نجيب الوقت بتاع ال algorithm وهو شغال على داتا بشكل معين وب size معين .. وعشان كدة عملوا definition لحاجة جديدة بتوصفلنا تقريبا كدة الوقت بتاع ال algorithm دة للداتا دي هيكون في range كام .. والحاجة دي اسمها algorithm time complexity
يارب تكون أهمية ال complexity كدة وضحت

يبقى معنى الكلام دة ان احنا محتاجين نبص على كل algorithm من 3 اتجاهات حتى لو كنت شغال ب data of the same size **(خلي بالك كل السيناريوهات اللي هنشوفها دلوقتي دي احنا مثبتين فيها ال size وبنغير الداتا بس)** :

1- Worst case scenario:
دة السيناريو اللي لو حصل فال algorithm هياخد اكبر وقت ممكن (for the same size) .. زي مثلا اني ابقى عايز ارتب list من الصغير للكبير بس هي اصلا جايالي مترتبة من الكبير للصغير

2- average case scenario:
دة بيعبر عن ال average time اللي ال algorithm بياخده عشان يشتغل على random list

3-Best case scenario:
اسرع وقت ال algorithm يخلص فيه .. زي اني ارتب list مترتبة اصلا

فعلى اساس ال size وال inputs بتاعتنا اللي في الداتا بنحسب ال complexity بتاعت كل algorithm ونختار ال algorithm المناسب للداتا دي ... وطبعا واضح ان كل الهري دة مالوش اي لازمة لو احنا بنشتغل على data صغيرة فيها 10 او 20 او 100 elements لأننا مش هنحس خالص بالفرق بين كل algorithm والتاني ... **فالكلام دة ميهمناش خالص غير لما نبدأ نكبر ال data size اللي بنشتغل عليها**

طب ال complexity بقى بنعبر عنها بإيه ؟؟ .. عن طريق حاجة في الرياضة اسمها big O notation (مش محتاجين نعرف عنها اي حاجة بس لو عايز تعرف عنها اكتر بص في ويكيبيديا)
فهنلاقي ان ال complexities بتاعتنا بنعبر عنها برموز ماسونية زي دي : $O(n), O(n \log n), O(n^\alpha), O(2^n)$

تعالى بس الاول نشرح الرموز دي جت ازاي وبعد كدة نشوف معناها ايه وبنستفيد منها بإيه :
دلوقتي يا سيدي انت عرفت بطريقةٍ ما توصف ال number of processed elements اللي عندك ب function زي كدة:
$f(n) = 5n^2+3n+4$

Number of processed elements

زي ما اتفقنا احنا مش عايزين نجيب exact value عشان الموضوع صعب .. احنا بس عايزين حاجة تقريبية نقدر نقارن بيها بين كل algorithm والتاني لما ال size يكبر .. فهنبص على ال dominant term في المعادلة لما n تكبر اوي … اللي هو هنا n^2 (ومماليش دعوة بال constant اللي جنبها في المعادلة عشان مش بحسب exact .. انا اهتمامي كله بالشكل اللي كفاءة ال algorithm بتتغير بيه لما ازود ال size)
فهناخد ال n^2 دي ونحطها في ال Big O بتاعتنا وتبقى دي ال time complexity بتاعت ال function دي : $O(n^2)$
ودي معناها ان التغير في وقت التنفيذ (او بمعنى تاني: التغير في ال algorithm efficiency) مع ال data size بيكون Quadratic في حين اننا لو جبنا algorithm تاني وعملناله نفس الكلام دة ولقينا انه O(n) يبقى معنى كدة ان ال algorithm التاني دة التغير فيه مع ال time بيكون linearمع ال size وبالتالي هو احسن لي من الاولاني

تعالى بقى نشوف امثلة على ال complexities :

O(1) :
دة بيقولي ان ال function مش بتعتمد على ال size .. فمهما كبرت ال size ال function هتاخد نفس الوقت عشان تتنفذ … زي مثلا ما بجيب element من array .. لو هجيب element من array فيها 10 elements ال processor هياخد نفس الوقت اللي هيجيب فيه element من array فيها مليون element

O(n) :
زي لما بجيب maximum او minimum item في unsorted list .. وبرضه دة ال order بتاع ال linear search

$O(n^2)$:
دة ال order بتاع selection sort & bubble sort & insertion sort

O(n log n) :
دة اسرع واحد وهو ال order بتاع حاجة اسمها comparison sort .. مخدناهوش ومسمعناش عنه حاجة بس يعني ربنا وحده اللي يعلم بالمادة دي واللي بنشوفه فيها :D

O(log n) :
دة ال order بتاع ال binary search

ودة اللينك اللي فيه كل الكلام دة :
https://en.wikipedia.org/wiki/Time_complexity

نرجع للمنهج بتاعنا بقى :)
احنا عملنا كل اللي فوق دة عشان نكمل الخواطر اللي بدأناها من شوية

لو جينا نقارن ال selection sort بال bubble sort (من غير ال early stop condition) وهم الاتنين بيشتغلوا على نفس الداتا وبنفس ال size (ال 2 examples اللي في السلايدز واللي احنا شرحنا عليهم فوق) وهنقارن من ناحية عدد ال comparisons اللي بتحصل ..
سلايد 59

Selection and Bubblesort comparison

- Both are $O(n^2)$ sorts
- If we count up the number of critical operations for both sorts, handling n-1, n-2, etc. pairs for each pass, using the data in the last example, we get
- Selection sort: 10 comparisons
- Bubble sort: 10 comparisons

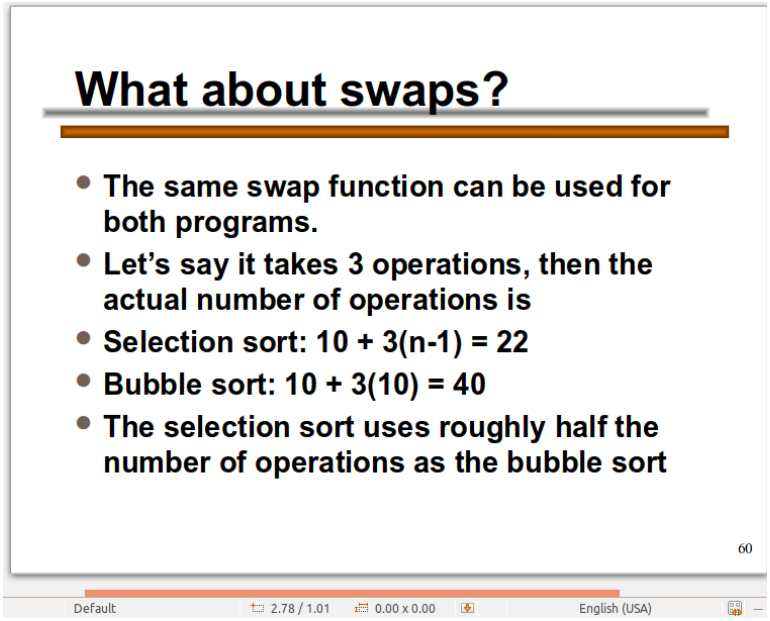
طب جيبنا ال 10 بتاعت ال selection دي منين؟؟
ارجع ل slide 35 :

Analysis: comparisons

- In the first pass we compared n-1 pairs
- In the second, n-2
- In the third, n-3, etc.
- Actual number of comparisons made across all passes, for this example was:
- 4+3+2+1 = 10

فكدة لما نبص على ال 2 algorithms من ناحية ال comparison هنلاقي ان هم الاتنين واحد ..
((لاحظ ان احنا هنا بنشتغل بأرقام مش ب big O ودة عشان ال lists صغيرة فمش مستاهلة التعقيد اللي كان فوق))
ممكن بقى نزود ال number of iterations كمان معانا في ال comparison (ماهي تعتبر operation برضه)

طب بالنسبة لل swaps ؟؟ .. ندخلها في المقارنة هي كمان مع ال comparisons (سلايد 60) :



هنا احنا بنجمع عشان نجيب ال operation بتاعت كل algorithm احنا بنجمع حاجتين على بعض، اللي هم ال comparisons & swaps
بس احنا عارفين من اولى ابتدائي ان مينفعش نجمع حاجتين مختلفين عن بعض .. فلازم نوصف حاجة بدلالة التانية .. هنا وصفنا ال swaps بدلالة ال comparison وفرضنا ان ال :

1 swap = 3 comparisons

فعرفنا نجمعهم على بعض .. ومن هنا قدرنا نعرف ان ال selection sort احسن لنا **(في الحالة دي)** من ال bubble sort

كدة المحاضرة خلصت وفاضل بس ال insertion sort اللي الدكتور مشرحوش
https://commons.wikimedia.org/wiki/File:Insertion-sort-example.gif
http://www.cs.armstrong.edu/liang/animation/web/InsertionSort.html
https://www.toptal.com/developers/sorting-algorithms/insertion-sort

---

النعيم لا يُدرك بالنعيم

ومن آثر الراحة فاتته الراحة