CMPT 120: Introduction to Computing Science and Programming 1

Sorting

Copyright © 2018, Liaqat Ali. Based on CMPT 120 Study Guide and Think Python - How to Think Like a Computer Scientist, mainly. Some content may have been adapted from earlier course offerings by Diana Cukierman, Anne Laveln, and Angelica Lim. Copyrights © to respective instructors. Icons copyright © to their respective owners.
How Fast is my Algorithm?

• There can be many algorithms to solve any problem – like linear search, binary search.

1. How do we choose the most efficient?
2. What is efficient?

• One measure is **how fast** our algorithm can determine the solution.
  ▫ This is not the only measure, nor is it always the best measure.
  ▫ How do we measure ‘how fast’.
Introduction to Sorting

- **Sorting**: Arranging values into an order:
  - Alphabetical
  - Ascending numeric
  - Descending numeric

- There are many problems that can be solved quite quickly by first sorting the values – like Binary searching.

- There are many algorithms to taking a list of elements and putting those elements into ascending or descending order.

- One of the simplest algorithms is **Selection sort**.
### Sorting a List: Ascending or Descending Order

**Unsorted List of size 5:**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>12</td>
<td>10</td>
<td>15</td>
<td>2</td>
</tr>
</tbody>
</table>

**Sorted List: Descending order**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>15</td>
<td>12</td>
<td>10</td>
<td>2</td>
</tr>
</tbody>
</table>

**Sorted List: Ascending order**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10</td>
<td>12</td>
<td>15</td>
<td>20</td>
</tr>
</tbody>
</table>
Selection Sort Example (Ascending Order)

**Iteration 1:**
1. Find the smallest element between lis[0] and lis[4].
2. Swap if smaller.

**Iteration 2:**
1. Find the smallest element between lis[1] and lis[4].
2. Swap if smaller.

**Iteration 3:**
1. Find the smallest element between lis[2] and lis[4].
2. Swap if smaller.

**Iteration 4:**
1. Find the smallest element between lis[3] and lis[4].
2. Swap if smaller.
Selection Sort

- A method to arrange the unsorted list into a sorted list.
- The algorithm sorts an list by repeatedly:
  - finding the **minimum** element from unsorted part and putting it at the beginning (for ascending order.)
  - finding the **maximum** element from unsorted part and putting it at the beginning (for descending order).
- Imagine the algorithm maintains two subarrays.
  1. The subarray which is already sorted.
  2. Remaining subarray which is unsorted.
- In every iteration, the minimum/maximum element from the unsorted subarray is picked and moved to the sorted subarray.
- Size of unsorted array gradually decreases as shown on next slide.
Selection Sort: Unsorted Array Size

Size = 5 = M
UnSorSize = N = 5 = M

M = 5
N = 4 = M - 1

M = 5
N = 3 = M - 2

M = 5
N = 2 = M - 3
Selection Sort - Procedure

1. Selection sort algorithm **starts by** comparing first two elements of an array and swapping, if necessary.

2. Then, again **first** element and **third** element are compared and swapped if necessary.

3. This process goes on until **first** and **last** element of an array is compared.

4. If there are \( N \) elements to be sorted then, the process mentioned above should be repeated \( N-1 \) times to get required result.

5. Finding the next lowest element requires scanning the remaining \( n - 1 \) elements and so on:

\[
(n - 1) + (n - 2) + \ldots + 2 + 1 = n(n - 1) / 2 = (n^2 - n) / 2 = \Theta(n^2) \text{ comparisons.}
\]
Selection Sort – Algorithm

for every element e from the list,

    for every element f from e to the end of the list,
        if f < smallest ,
            set smallest to f
    swap smallest and e
Canvas Post: Due by Monday, July 30, 11:59pm

- Implement the **selection sort** algorithm. You may refer to the course Study Guide *Figure 6.5* and *Figure 6.6* for selection sort algorithm and its implementation.

- Use the following list to test your program:
  
  \[1, 6, 0, 7, 10\]

- Post your Python program and its result on Canvas.
Questions?