Sorting: Putting your affairs in order

Nelly Sin

October 2021

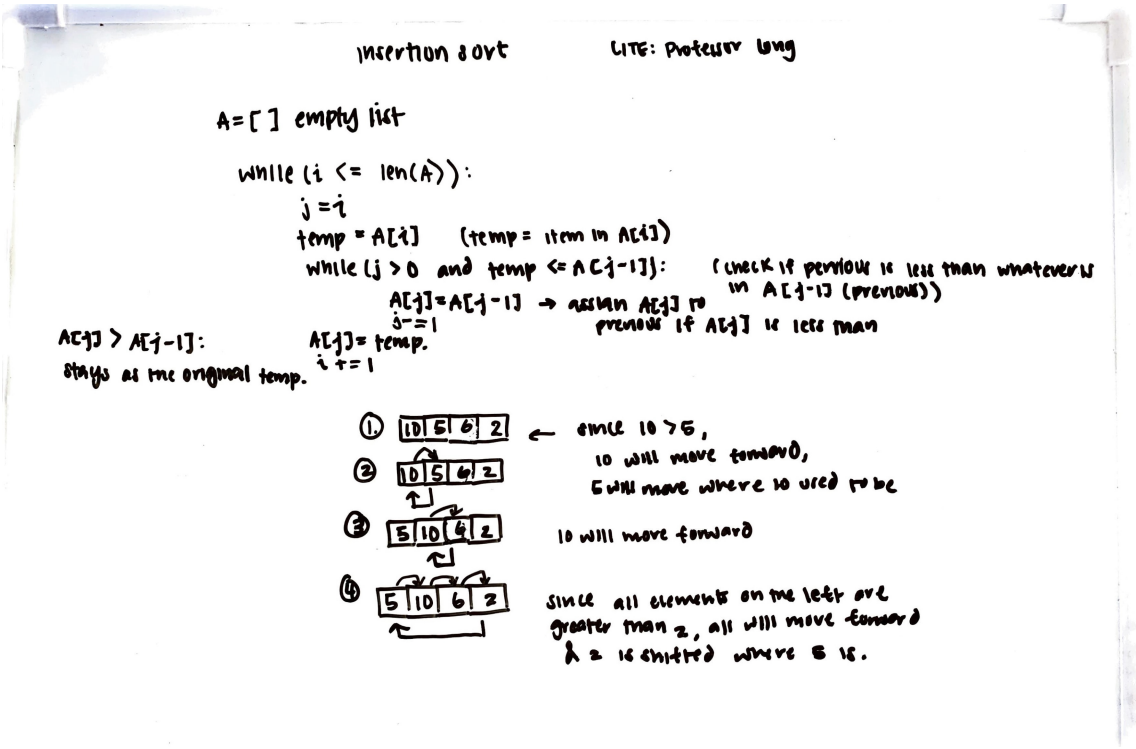
1 Insertion Sort

Cite: Professor Long for Python sudo Code

Insertion Sort in Python

```
1 def insertion_sort(A: list):
2     for i in range(1, len(A)):
3         j = i
4         temp = A[i]
5         while j > 0 and temp < A[j - 1]:
6             A[j] = A[j - 1]
7             j -= 1
8         A[j] = temp
```

Insertion sort is one probably the most basic sort or "go-to" sort. It compares the next and previous elements and rearrange them as they go.



Cite: Eugene for explanation This is the visualization for insertion sort.

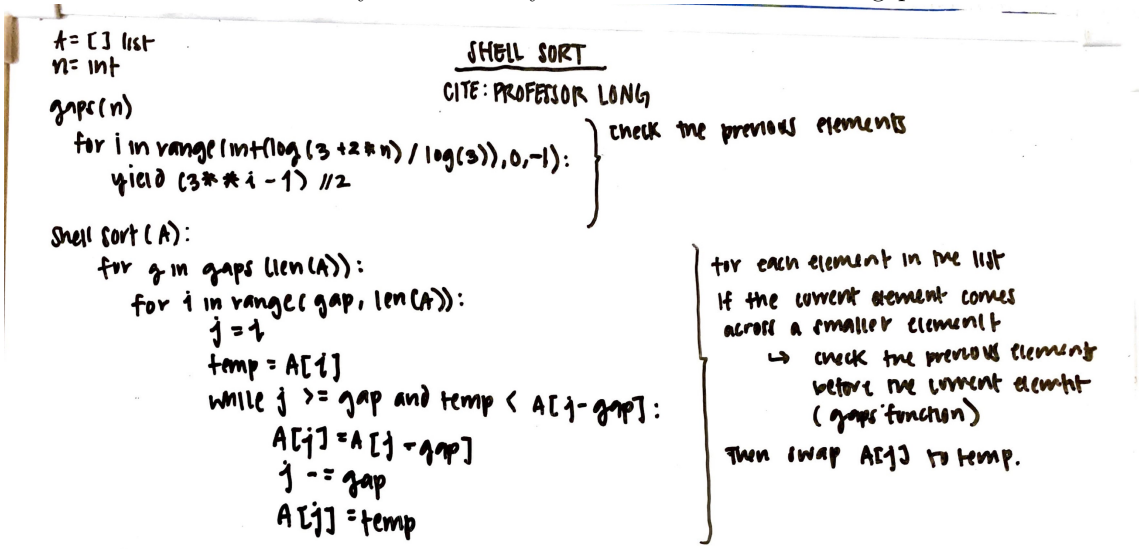
2 Shell Sort

Cite: Professor Long for Python sudo Code

Shell Sort in Python

```python
from math import log

def gaps(n: int):
    for i in range(int(log(3 + 2 * n) / log(3)), 0, -1):
        yield (3**i - 1) // 2

def shell_sort(A: list):
    for gap in gaps(len(A)):
        for i in range(gap, len(A)):
            j = i
            temp = A[i]
            while j >= gap and temp < A[j - gap]:
                A[j] = A[j - gap]
                j -= gap
            A[j] = temp
```

Shell sort relies on the gap and compare specific elements in the of the array. Such as every second element – this is the gap.



Cite: Eugene for explanation This is the visualization of shell sort. Your can see there is a gap between each element when sorting.

3 Heap Sort

Cite: Professor Long for sudo code

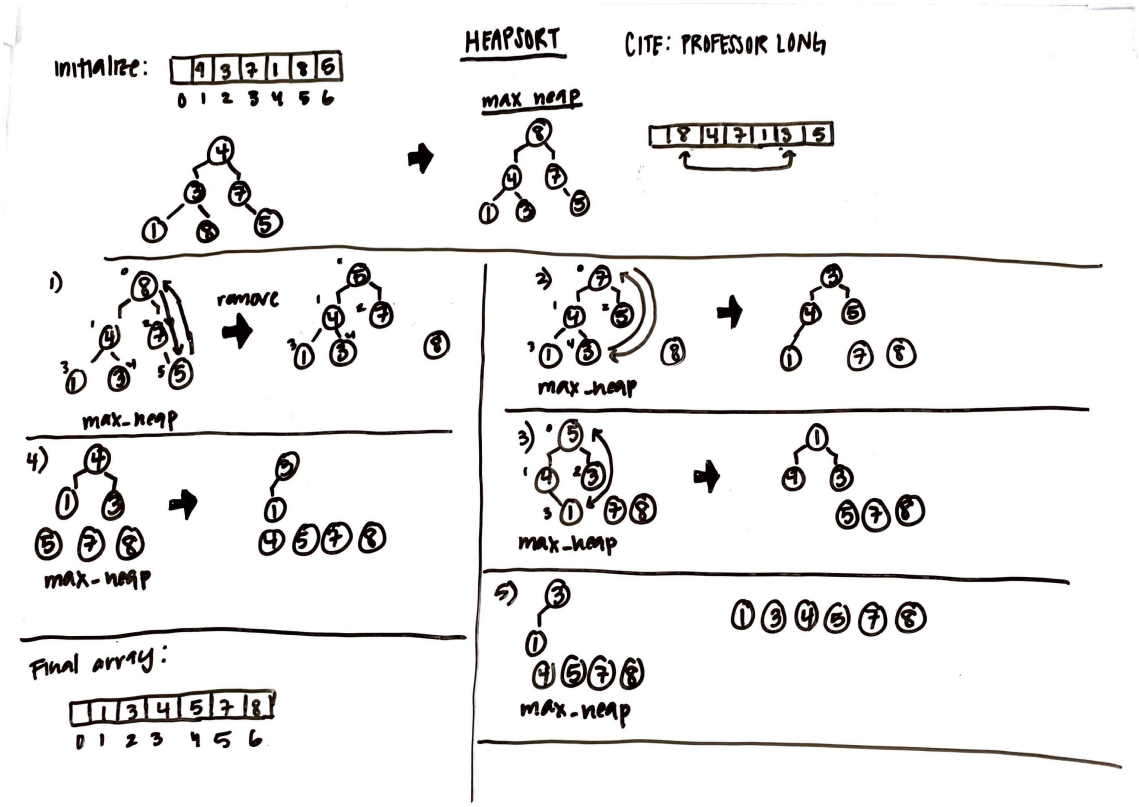
Heap maintenance in Python

```python
def max_child(A: list, first: int, last: int):
    left = 2 * first
    right = left + 1
    if right <= last and A[right - 1] > A[left - 1]:
        return right
    return left

def fix_heap(A: list, first: int, last: int):
    found = False
    mother = first
    great = max_child(A, mother, last)

    while mother <= last // 2 and not found:
        if A[mother - 1] < A[great - 1]:
            A[mother - 1], A[great - 1] = A[great - 1], A[mother - 1]
            mother = great
            great = max_child(A, mother, last)
        else:
            found = True
```

Heapsort in Python

```python
def build_heap(A: list, first: int, last: int):
    for father in range(last // 2, first - 1, -1):
        fix_heap(A, father, last)

def heap_sort(A: list):
    first = 1
    last = len(A)
    build_heap(A, first, last)
    for leaf in range(last, first, -1):
        A[first - 1], A[leaf - 1] = A[leaf - 1], A[first - 1]
        fix_heap(A, first, leaf - 1)
```

Heap, I think is one of the most difficult to visualize. Since there are rules in the heap sort. Such as having parent elements that will always be larger than the children and sorting the children to have the max child – max heap. After finding the largest element, the element will be removed from the beginning of the array to the end of the array.



Cite: Eugene for explanation This is the visualization of heap sort. Drawing a tree is the most simple way to explain it since it's similar to a family tree. Where the oldest is on the top of the tree and following are suppose to be the youngest.

4 Quick Sort

Cite: Professor Long for sudo code

Partition in Python

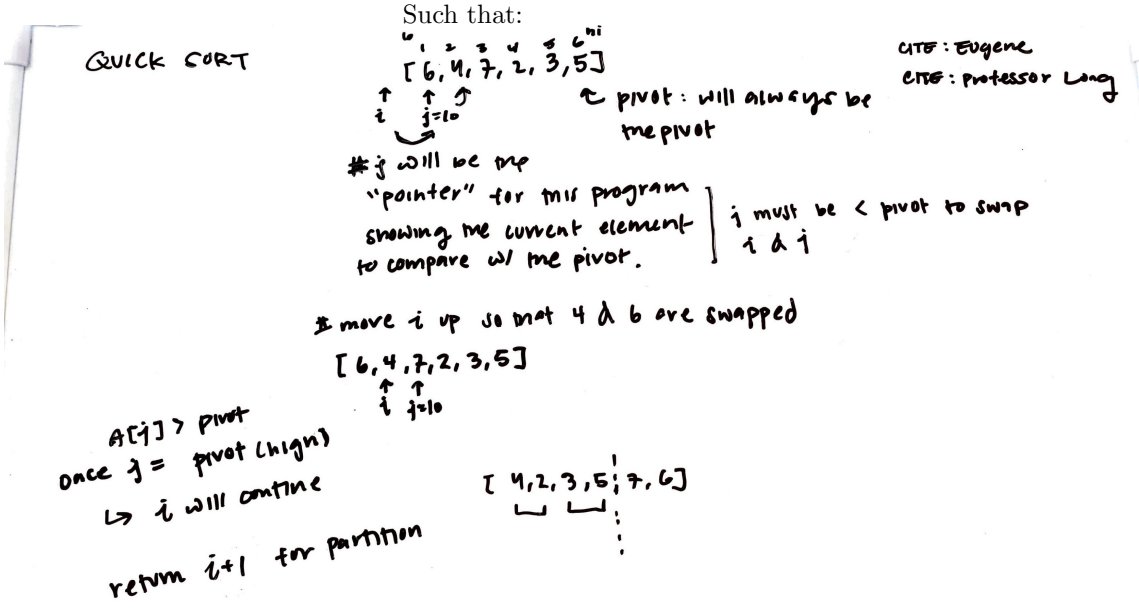
```python
def partition(A: list, lo: int, hi: int):
    i = lo - 1
    for j in range(lo, hi):
        if A[j - 1] < A[hi - 1]:
            i += 1
            A[i - 1], A[j - 1] = A[j - 1], A[i - 1]
    A[i], A[hi - 1] = A[hi - 1], A[i]
    return i + 1
```

Recursive Quicksort in Python

```python
# A recursive helper function for Quicksort.
def quick_sorter(A: list, lo: int, hi: int):
    if lo < hi:
        p = partition(A, lo, hi)
        quick_sorter(A, lo, p - 1)
        quick_sorter(A, p + 1, hi)

def quick_sort(A: list):
    quick_sorter(A, 1, len(A))
```

Quick sort focus on one part of the array (partitions). Each partition has a pivot, low, and high where the pivot compare each element in the array, low and high are the ones that are being compared. If high is smaller than the pivot, the high will switch with the low.

Such that:



Cite: Eugene for explanation As seen here, the pivot can be anywhere – doesn't exaclty have to be in the middle. i and j are the low and high I was explaining earlier.

5 Testing Harness

Cite: Professor Long, Eugene, and Sloan for explanation

The center holds all the command lines, functions, and the statistics of all the sort. It may be the most repetitive since there are a lot of printing and obtaining user input for some of the command lines.
Here is what I have in mind when designing my main function. There are certain variables that must be set to a default and obtaining a list of random arrays.