

Assignment 3: Sorting: Putting your affairs in order

Kelly Liu

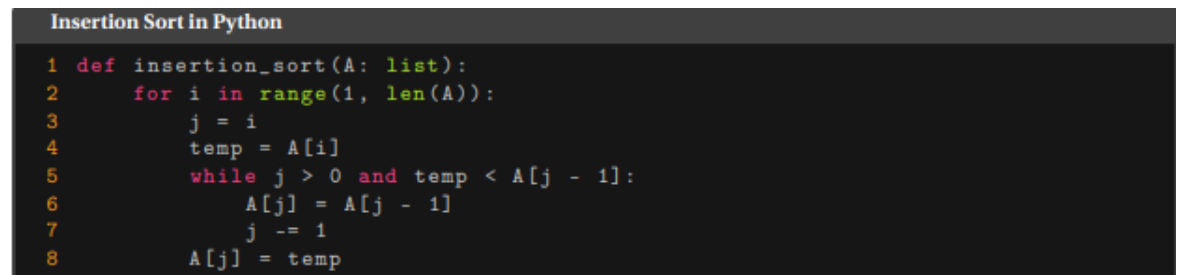
October 14, 2021

1 Description

In this program, we are making a bunch of different sort functions using the following methods: Insertion, Shell Sort, Heap Sort and Quick Sort. We will also be using a test file to test our sorting algorithms using pseudorandom elements and then gathering statistics based on the results.

2 Pseudocode

2.1 insertion.c



```
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
```

The picture above gives me the basic pseudocode for my insertion sort function. However, there are modifications that I need to do in order to meet the requirements of the assignment. I need to include all files first. Then, I'm expecting to pass 2 other arguments, how long the array is and stats, along with the list. I'm also expecting to be using my stats.c source code, specifically the cmp and move functions. while I'm coding the sort function.

2.2 shell.c

```
Shell Sort in Python
1 from math import log
2
3 def gaps(n: int):
4     for i in range(int(log(3 + 2 * n) / log(3)), 0, -1):
5         yield (3*i - 1) // 2
6
7 def shell_sort(A: list):
8     for gap in gaps(len(A)):
9         for i in range(gap, len(A)):
10             j = i
11             temp = A[i]
12             while j >= gap and temp < A[j - gap]:
13                 A[j] = A[j - gap]
14                 j -= gap
15             A[j] = temp
```

The picture above gives me the basic pseudocode for my shell sort function. However, there are modifications that I need to do in order to meet the requirements of the assignment. This one should be similar to insertion.c but with a different sorting technique. I need to include all files first. Then, I'm expecting to pass 2 other arguments, how long the array is and stats, along with the list. I'm also expecting to be using my stats.c source code, specifically the cmp and move functions. while I'm coding the sort function.

2.3 heap.c

Heap maintenance in Python

```
1 def max_child(A: list, first: int, last: int):
2     left = 2 * first
3     right = left + 1
4     if right <= last and A[right - 1] > A[left - 1]:
5         return right
6     return left
7
8 def fix_heap(A: list, first: int, last: int):
9     found = False
10    mother = first
11    great = max_child(A, mother, last)
12
13    while mother <= last // 2 and not found:
14        if A[mother - 1] < A[great - 1]:
15            A[mother - 1], A[great - 1] = A[great - 1], A[mother - 1]
16            mother = great
17            great = max_child(A, mother, last)
18        else:
19            found = True
```

Heapsort in Python

```
1 def build_heap(A: list, first: int, last: int):
2     for father in range(last // 2, first - 1, -1):
3         fix_heap(A, father, last)
4
5 def heap_sort(A: list):
6     first = 1
7     last = len(A)
8     build_heap(A, first, last)
9     for leaf in range(last, first, -1):
10        A[first - 1], A[leaf - 1] = A[leaf - 1], A[first - 1]
11        fix_heap(A, first, leaf - 1)
```

The picture above gives me the basic pseudocode for my heap sort function. However, there are modifications that I need to do in order to meet the requirements of the assignment. This sort is different from the other two because I need to be able to create a heap before I start sorting. The pseudocode for this is also provided in the image above. I need to include all files first. Then, I'm expecting to pass 2 other arguments, how long the array is and stats, along with the list. I'm also expecting to be using my stats.c source code, specifically the cmp and swap functions. while I'm coding the sort function.

2.4 quick.c

Partition in Python

```
1 def partition(A: list, lo: int, hi: int):
2     i = lo - 1
3     for j in range(lo, hi):
4         if A[j] < A[hi]:
5             i += 1
6             A[i], A[j] = A[j], A[i]
7     A[i], A[hi] = A[hi], A[i]
8     return i + 1
```

Recursive Quicksort in Python

```
1 # A recursive helper function for Quicksort.
2 def quick_sorter(A: list, lo: int, hi: int):
3     if lo < hi:
4         p = partition(A, lo, hi)
5         quick_sorter(A, lo, p - 1)
6         quick_sorter(A, p + 1, hi)
7
8 def quick_sort(A: list):
9     quick_sorter(A, 0, len(A) - 1)
```

The picture above gives me the basic pseudocode for my quick sort function. However, there are modifications that I need to do in order to meet the requirements of the assignment. This one should be similar to `heapsort.c` but with a different sorting technique. This differs from the previous three functions because I would need to make my partition function before I start sorting. The pseudocode for this is also provided in the image above. I need to include all files first. Then, I'm expecting to pass 2 other arguments, how long the array is and stats, along with the list. I'm also expecting to be using my `stats.c` source code, specifically the `cmp` and `swap` functions while I'm coding the sort function.

2.5 notes

Most of the pseudocode I had provided in my first draft of my design document worked as plan so I didn't feel the need to add anything onto it in my final draft. I basically followed the python pseudocode provided by Professor Long to the letter, with the addition of adding extra arguments and using the functions provided in my stats files.

The most notable change was with my shell function because C does not have anything similar to `yield` in Python. I bypassed this by doing my gap computation within my shell sort function. By implementing a triple for loop, I didn't need to create a gap function.

2.6 sorting.c

This file contains the main function that will run the program and execute the sorting that the user inputted. I need to parse the user input using what I did in lab 2 and output the results like what the example binary would.

I used a set to determine which sorts to run. Similar to the last lab, I used getopt to get the arguments from the command. Then, I added the respective sorting to the set. Similarly, if a seed, size, or element size is specified, then they will be stored to the respective variables in preparation for other sorts. If there are no more arguments, then the program will run the sorts and print out the outputs. In a case for no input or help, then the help message will be printed.

3 Files

insert.c- A source file for sorting a pseudorandom array using insertion function.

insert.h- A header file that specifies the interface for insert.c.

heap.c- A source file for sorting a pseudorandom array using heap sort.

heap.h- A header file that specifies the interface for heap.c.

quick.c- A source file for sorting a pseudonym array using quicksort.

quick.h- A header file that specifies the interface for quick.c.

shell.c- A source file for sorting a pseudorandom array using shell sort.

shell.h- A header file that specifies the interface for shell.c.

set.c- A source file that implements and specifies the interface for set ADT.

stats.c- A source file that has my statistics module.

sorting.c- A source file that contains my main function.

Makefile- This allows us to use clang and compile our program.

README.md- In markdown format, it tells us how to run the program and how the program was made.

DESIGN.pdf- This is how I started thinking about how to code the program.

WRITEUP.pdf- A PDF considering graphs made by UNIX tool displaying my results and my thoughts on them.

4 Credit

1. Professor Long has provided python pseudocode in the Assignment 3 description PDF for our sorting functions that I have used for inspiration.

2. Professor Long has provided us with a few files in the resources folder in Assignment 3.

3. Professor Long provided me with help on how to do gnuplot in his lecture video.

4. I used Eugene's bubble sort example to help me write my sorting algorithms.

5 Error

1. Like always, I had a lot of syntax errors that were easily fixed whenever I compiled and saw the errors that came up.
2. I didn't use the move function correctly provided in our stats file.
3. I had issues implementing the yield function that Python has in C, and I struggled a bit before realizing I can bypass using yield if I just made my gap function within my shell sort function.
4. I had a segmentation error in my shell function because I had used the wrong variable while iterating through my for loop.
5. My sorting.c file has the most issues, I had undeclared variables, lost variables, and memory issues. This was all debugged once compiled after carefully looking through my code.