Jaden Liu

Zliu259@ucsc.edu

11/13/2020

CSE13s Fall 2020

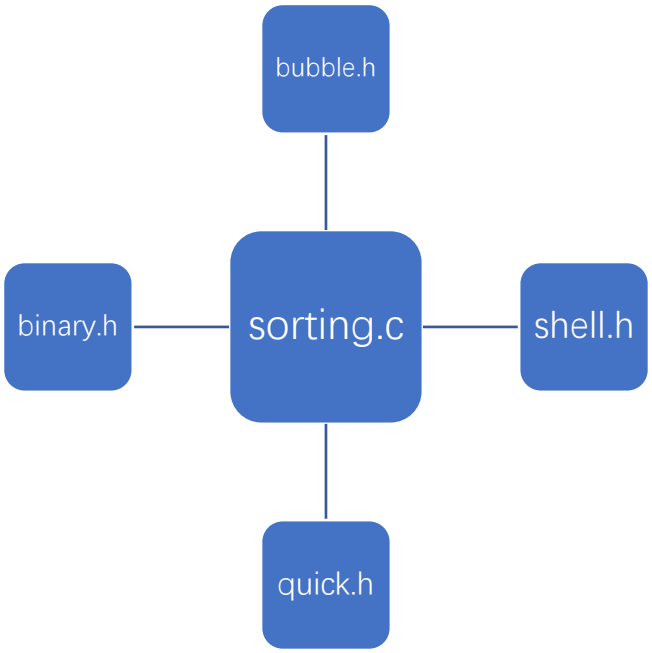
Assignment5: Sorting

Design document

In this assignment, I implement 4 types of sorting algorithm, which are Bubble sort, Shell sort, Quick sort and Binary Insertion sort. I will implement each sorting with a separate .c and .h file to implement them in the main file sorting.c.

I answered each pre-lab question in each page because it is better to answer them in their own algorithm part.

Design chart for this assignment:



Sorting.c

In this file, it contains the main function to get command from command line by the function: getopt().

-A means employ all sorting algorithms.

-b means enable Bubble Sort.

-s means enable Shell Sort.

-q means enable QuickSort.

-i means enable Binary Insertion Sort.

-p n means print the first n elements of the array. However if the -p n flag is not specified, your program should print the first 100 elements. The default n value is 100.

-r s means set the random seed to s. The default s value is 8222022.

-n c means set the array size to c. The default c value is 100.

```
int main(int argc, char **argv){
    int c=0;
    while ((c = getopt (argc , argv , " Absqip :r:n:")) != -1) {
        switch(c){
            case “A”:
                print all sorting method
                break
            case “b”:
                print bubble sort
                break
            case “s”:
                print shell sort
                break
            case “q”:
                print quick sort
                break
            case “i”:
                print binary insertion sort
                break
            case “p”:
                print first n element in the array
                break
            case “r”:
                set random seed to s, default is 8222022
                break
            case “n”:
                set array size to c, default is 100
        }
    }
}
```

Bubble sort

Pseudocode (cited from asgn5.pdf):

```
def Bubble_Sort(arr):
  for i in range(len(arr) - 1):
    j = len(arr) - 1
    while j > i:
      if arr[j] < arr[j - 1]:
        arr[j], arr[j - 1] = arr[j - 1], arr[j]
      j -= 1
  return
```

Pre-lab Part 1

1. How many rounds of swapping do you think you will need to sort the numbers 8, 22, 7, 9, 31, 5, 13 in ascending order using Bubble Sort?
2. How many comparisons can we expect to see in the worse case scenario for Bubble Sort? Hint: make a list of numbers and attempt to sort them using Bubble Sort.

1. 8,22,7,9,31,5,13
   8,22,7,9,31,5,13 +1
   8,22,7,9,5,31,13 +1
   8,22,7,5,9,31,13 +1
   8,22,5,7,9,31,13 +1
   8,5,22,7,9,31,13 +1
   5,8,22,7,9,31,13 +1
   5,8,22,7,9,13,31
   5,8,22,7,9,13,31
   5,8,22,7,9,13,31 +1
   5,8,7,22,9,13,31 +1
   5,7,8,22,9,13,31
   5,7,8,22,9,13,31
   5,7,8,22,9,13,31 +1
   5,7,8,9,22,13,31
   5,7,8,9,22,13,31
   5,7,8,9,22,13,31 +1
   5,7,8,9,13,22,31
   5,7,8,9,13,22,31
   5,7,8,9,13,22,31
   10 times of swapping in total.
2. If there are n numbers, the worst case is doing n(n+1)/2 Times of comparisons.

Shell sort

Pseudocode (cited from asgn5.pdf):

```
def gap(n):
  while n > 1:
    n = 1 if n <= 2 else 5 * n // 11
    yield n
```

gap (pseudocode)

```
def Shell_Sort(arr):
  for step in gap(len(arr)):
    for i in range(step, len(arr)):


      for j in range(i, step - 1, -step):
        if arr[j] < arr[j - step]:
          arr[j], arr[j - step] = arr[j - step], arr[j]
  return
```

Shell Sort (pseudocode)

Pre-lab Part 2

1. The worst time complexity for Shell sort depends on the size of the gap. Investigate why this is the case. How can you improve the time complexity of this sort by changing the gap size? Cite any sources you used.
2. How would you improve the runtime of this sort without changing the gapp size?

1. The worst time complexity is set gap to 1, which is the same as insertion sort. Insertion sort is good to do almost sorted array. Therefore, it is better to compared in each interval first.
   The better gap sequence is like this:

$4^k + 3 \cdot 2^{k-1} + 1$, prefixed with 1	$1, 8, 23, 77, 281, \ldots$	$O\left(N^{\frac{4}{3}}\right)$

   Cited from:
   Sedgewick, Robert (1998). Algorithms in C. 1 (3rd ed.). Addison-Wesley. pp. 273–281. ISBN 978-0-201-31452-6.
2. Since Shell sort is just a variation of standard insertion sort, it can be optimized by the same way.
   Before doing the sorting algorithm, compare each paired numbers and put the larger one on the right(ascending sequence). Therefore, it can save half of time to visit the array, since half of the big has already on the right.
   Pseudocode:

```
        for (int index = length - 1; index > 0; index--) {
            if (less(arr[index], arr[index - 1])) {
                exch(arr, index, index - 1);
                exchanges++;
            }
        }
```

Cited from: https://blog.csdn.net/tinydolphin/article/details/72773494

Quick sort

Pseudocode (cited from asgn5.pdf):

```
def Partition(arr, left, right):
  pivot = arr[left]
  lo = left + 1
  hi = right

  while True:
    while lo <= hi and arr[hi] >= pivot:
      hi -= 1

    while lo <= hi and arr[lo] <= pivot:
      lo += 1

    if lo <= hi:
      arr[lo], arr[hi] = arr[hi], arr[lo]
    else:
      break

  arr[left], arr[hi] = arr[hi], arr[left]
  return hi


def Quick_Sort(arr, left, right):
  if left < right:
    index = Partition(arr, left, right)
    Quick_Sort(arr, left, index - 1)
    Quick_Sort(arr, index + 1, right)
  return
```

Pre-lab Part 3

1. Quicksort, with a worse case time complexity of $O(n^2)$, doesn't seem to live up to its name. Investigate and explain why Quicksort isn't doomed by its worst case scenario. Make sure to cite any sources you use.

1. It is crucial to identify the first pivot. If the first pivot in the middle point, it is usually the average time complexity, which is O(nlogn). While if it choose the first value, then it has the worst time complexity O(n^2)

Binary Insertion Sort

Pseudocode (cited from asgn5.pdf):

```
def Binary_Insertion_Sort(arr):
  for i in range(1, len(arr)):
    value = arr[i]
    left = 0
    right = i

    while left < right:
      mid = left + ((right - left) // 2)

      if value >= arr[mid]:
        left = mid + 1
      else:
        right = mid

    for j in range(i, left, -1):

      arr[j - 1], arr[j] = arr[j], arr[j - 1]

  return
```

Binary Insertion Sort (pseudocode)

Pre-lab Part 4

1. Can you figure out what effect the binary search algorithm has on the complexity when it is combined with the insertion sort algorithm?

1. The worst time complexity for insertion sort algorithm is O(n^2). With the help of binary search algorithm, it can improve to O(nlogn), since one of the O(n) improves to O(logn) because of the binary algorithm.

Pre-lab Part 5

1. Explain how you plan on keeping track of the number of moves and comparisons since each sort will reside within its own file.

1. I plan to use global value, since it won't extinct in and out of the function. So, it can be good to store the value for each sorting algorithms.

Design process

Over the course of this assignment, it is relatively smoothly to design this assignment since almost all pseudocode is provided.

All in all, from this assignment, I learned three extra sorting algorithms besides bubble sort and insertion sort. Also, I learned more about the time complexity and how to optimize the algorithm.