CS325\_HW1

Yihong Liu

1.

After simplification, we can just compute the function n = 8 log(2, n) , and we got the two results, n≈1.1n≈43.5593

And we found that while n is bigger than 1.1 but smaller than 43.5593, the insertion running time is smaller than merge, which is faster. So, the answer is [1.1 – 43.5593].

2.

a. O(g(n)), because the limit as n goes to inﬁnity of f(n) / g(n) is 0.

b. Ω(g(n)), because the limit as n goes to inﬁnity of f(n) / g(n) is infinity.

c. Θ(g(n)), because the limit as n goes to inﬁnity of f(n) / g(n) is a constant.

d. Θ(g(n)), because the limit as n goes to inﬁnity of f(n) / g(n) is a constant.

e. O(g(n)), because the limit as n goes to inﬁnity of f(n) / g(n) is 0.

f. O(g(n)), because the limit as n goes to inﬁnity of f(n) / g(n) is 0.

g. Θ(g(n)), because the limit as n goes to inﬁnity of f(n) / g(n) is a constant.

h. O(g(n)), because the limit as n goes to inﬁnity of f(n) / g(n) is 0.

i. Ω(g(n)), because the limit as n goes to inﬁnity of f(n) / g(n) is infinity.

j. O(g(n)), because the limit as n goes to inﬁnity of f(n) / g(n) is 0.

3.

a. Proves, because f1(n) =Θ (g(n)) so, f1(n)=c1g(n) and f2(n)= Θ (g(n))=c2g(n) we can get f1(n)/f2(n) is equally a Constant c so f1(n)= Θf2(n) is prove

b. disproves, if f1(n)=f2(n)=2^n g1(n)=g2(n)=20^n so, lim n->infinity, (f1(n)+f2(n))/(g1(n)+g2(n)) is 0. It’s O(g(n)) instead of Θ(g(n)), there is a contradiction.

4.

See teach files

5.

a.

**insertTime.py**

from random import randint

from time import time

MIN\_INT = 0

MAX\_INT = 10000

def insertsort(array):

#go through each element in the array

for j in range (1,len(array)):

key = array[j];

i = j - 1

while i >= 0 and array[i] > key:

array[i+1] = array[i]

i = i -1

array[i+1] = key

return array

if \_\_name\_\_ == "\_\_main\_\_":

begin = 10000

increment = 5000

values = [begin + i \* increment for i in range(7)]

list = [[randint(MIN\_INT,MAX\_INT) for \_ in range(value)] for value in values]

print ("value running time")

for array in list:

array\_size = len(array)

start\_time = time()

insertsort(array)

print("{} {}".format(array\_size,time()-start\_time))

**mergeTime.py**

from random import randint

from time import time

MIN\_INT = 0

MAX\_INT = 10000

def merge(left,right):

#create empty array to hold sorted values

array = []

#keep merge until one side has no element left

while len(left) != 0 and len(right) != 0:

#if the first element of left array is smaller merge it into empty array

#and remove the merged element

if left[0] < right[0]:

array.append(left[0])

left.remove(left[0])

# the situation that is equal or bigger than right side

else:

array.append(right[0])

right.remove(right[0])

#if the left side array is empty, then append the right side

if len(left) == 0:

array += right;

else:

array += left;

return array

#define mergesort algorithm

def mergesort(array):

array\_size = len(array)

if array\_size <= 1:

return array

else:

array\_split = array\_size//2

left\_array = mergesort(array[:array\_split])

right\_array = mergesort(array[array\_split:])

return merge(left\_array,right\_array)

if \_\_name\_\_ == "\_\_main\_\_":

begin = 10000

increment = 5000

values = [begin + i \* increment for i in range(7)]

list = [[randint(MIN\_INT,MAX\_INT) for \_ in range(value)] for value in values]

print ("value running time")

for array in list:

array\_size = len(array)

start\_time = time()

mergesort(array)

print("{} {}".format(array\_size,time()-start\_time))

b.

Here is the table result from my algorithms, I took three tables and got the average running time.

insertTime table

|  |  |
| --- | --- |
| N | averageT |
| 0 | 0 |
| 10000 | 6.9 |
| 15000 | 15.69 |
| 20000 | 27.78 |
| 25000 | 43.33 |
| 30000 | 62.93 |
| 35000 | 84.89 |
| 40000 | 110.91 |

mergeTime table

|  |  |  |
| --- | --- | --- |
| N | averageT |  |
| 0 | 0 |  |
| 10000 | 0.1578 |  |
| 15000 | 0.1857 |  |
| 20000 | 0.2593 |  |
| 25000 | 0.3531 |  |
| 30000 | 0.5042 |  |
| 35000 | 0.6367 |  |
| 40000 | 0.682 |  |

c.

I used the polynomial curve to fit the insertTime graph, r^2 is 1, which means it fits perfectly.

And I used linear curve to fit the mergeTime graph, r^2 is 0.9601, which is also considered as extreme good.

d.

The combined graph, the yellow line is mergeTime, the blue one is insertTime, and there are two Y-aixs.

e.

For the Insert Sort algorithm, when using the worst-case input, the experimental running times appeared to be very close to the theoretical worst case, O(n^2), as expected. The curve ﬁt for the above graph points to an equation of the f(n) = n^2 nature. When the best-case input was use, the experimental running times appeared to be very close to the theoretical best case, O(n), as expected.

For the Merge Sort algorithm, when using the worst-case input, the experimental running times appeared to be very close to O(n lgn), as expected. There technically isn’t a worst case for the Merge Sort algorithm, but it was discovered an array setup that would be the hardest to sort. See link at the bottom of mergesort\_worst.py The curve ﬁt for the above graph points to an equation of the f(n) = n nature. When the best-case input was use, the experimental running times appeared to be very close to O(n lgn), as expected. Again, there technically isn’t a best case for the Merge Sort algorithm but the results of this “best case” were faster than the “worst case”.