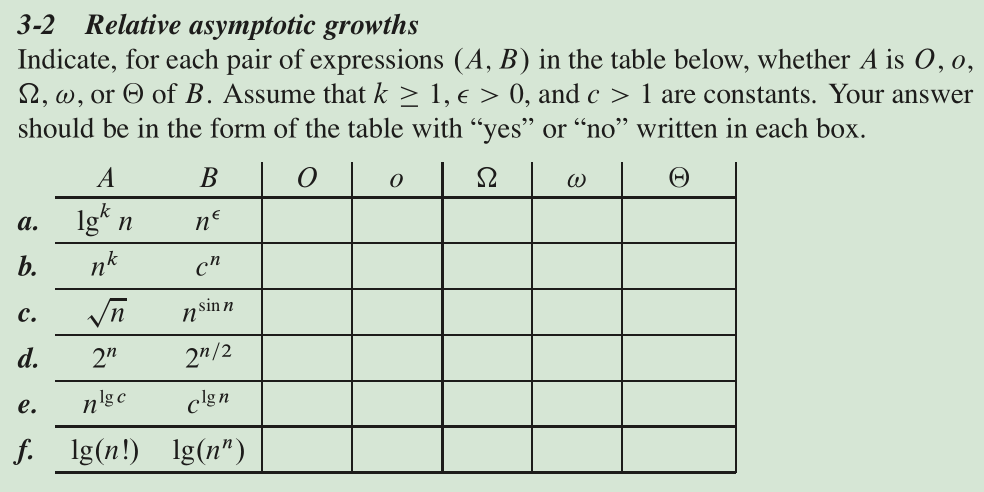
**EC 504 Data Structure**

**HW1**

**Name: Yangruirui Zhou BU ID：U16997747**

1. a)



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | O | o |  |  |  |
| a | y | y | n | n | n |
| b | y | y | n | n | n |
| c | n | n | n | n | n |
| d | n | n | y | y | n |
| e | y | n | y | n | y |
| f | y | n | y | n | y |

b)

(c) Because , and . We can get following equations:

,

(extra) From ,

Thus, it means only when n=0, the equation can be satisfied.

From ,

,

2.

(a) In each recursion, we need to execute the conditional statement to judge if n equals to 0 and complete the multiple operation A(n-1) \* A(n-1), so totally 2 executions. And in the last recursion, we will only execute 2 statements: first, is the conditional statement and the second one is “return 1”. Thus, we can have following equations:

From the structure, we can assume and substitute it into

Thus, we can substitute and into , and we can get

(b) In each recursion (n>1), we need to execute the conditional statement to judge if n equals to 0 and if and execute one return statement, so totally 3 normal execution + twice B(n/2) call. When n=1, we will only need to call B(n/2) once, because B(1/2) = B(0) = 1 < 10. Thus, we totally need 3 normal execution + once B(n/2) call. When n=0, we will only need two execution includes judging if n equals to 0 and return 1, so T(0)=2. Thus, we can have following equations:

(i) When n>1 and n is even number:

From the structure, we can assume

Thus, we can substitute and into

(ii) When n>1 and n is odd number:

From the structure, we can assume

Thus, we can substitute and into

So,

(c) In each recursion (n>1), we need to execute the conditional statement to judge if n equals to 0 and if n equals to 1 and execute one return statement, so totally 3 normal execution + once D(n-1) call +twice D(n-2) call. And it is very easy to know the number of executions when n equals to 0 and 1. So we can get following equations:

From the structure, we can assume

Thus, we can substitute and into we can find is not exist. So, the assumption fails, we have to analyze the borders of it.

(i) For the lower border, note that T(n) > (n>1), which can be calculate by:

Let T(n)= in the difference equation  to obtain , which yields the solutions and leads to the general form

. Assume the values =0 and =1 then

and T(n) =

(ii) For the upper border, note that T(n) < H(n) (n>1), which can be calculate by:

We can assume and =6

Thus,

From above, we can know

3. Pseudo-code description:

Get inputs from terminal, let nuts[] be the sets of nuts, bolts[] be the sets of bolts, n is their number. Swap() means value transition. PrintArray() means print two arrays on the console.

int find(char arr[], int low, int high, char midvalue, int flag)

// flag is used to remark the order of nut and bolt in TEST()

{

    int i = low;

    for(int j = low; j < high; j++)

{

if (flag > 0) a = arr[j]; b = midvalue;

else a = midvalue; b = arr[j];

// This is used to judge if midvalue is bigger than arr’s element.

// If it is true, I will put it to midvalue’s left.

if (TEST(a, b) \* flag == -1)

        {

Swap(arr[i],arr[j]);

            i++;

        }

        else if (TEST(a, b) == 0)

        {

Swap(arr[j],arr[high]);

            j--;

        }

}

Swap(arr[i],arr[high]);

    return i;

}

// Function which works just like quick sort, low is the smallest subtitle of both arrays, high is the biggest subtitle of both arrays.

void Qsort(char nuts[], char bolts[], int low, int high)

{

    if (low < high)

    {

        int middle = find(nuts,low, high, bolts[high], 1)

        find(bolts, low, high, nuts[middle], -1);

        Qsort(nuts, bolts, low, middle - 1);

        Qsort(nuts, bolts, middle + 1, high);

}

PrintArray (nuts, bolts)

}

4. Coding work report

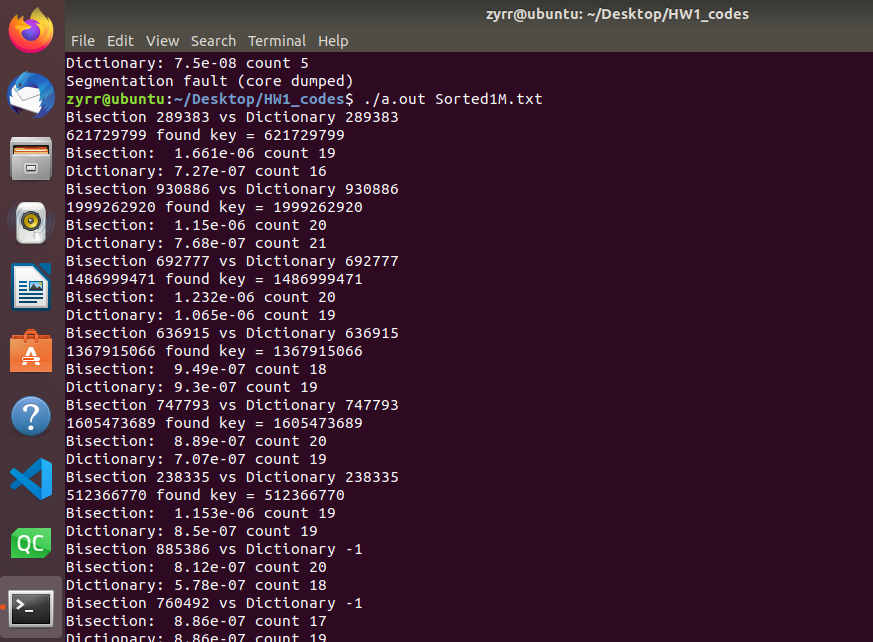


Fig1. Command line input and output

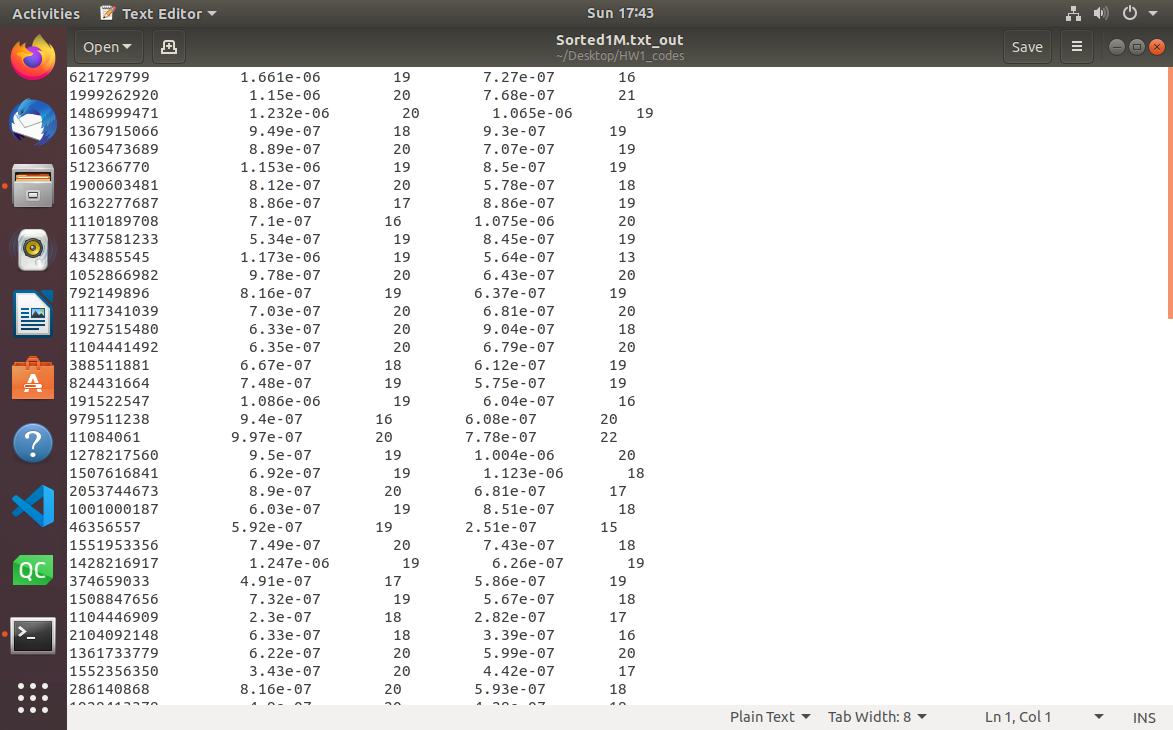


Fig2. Output file (Sorted1M.txt\_out)

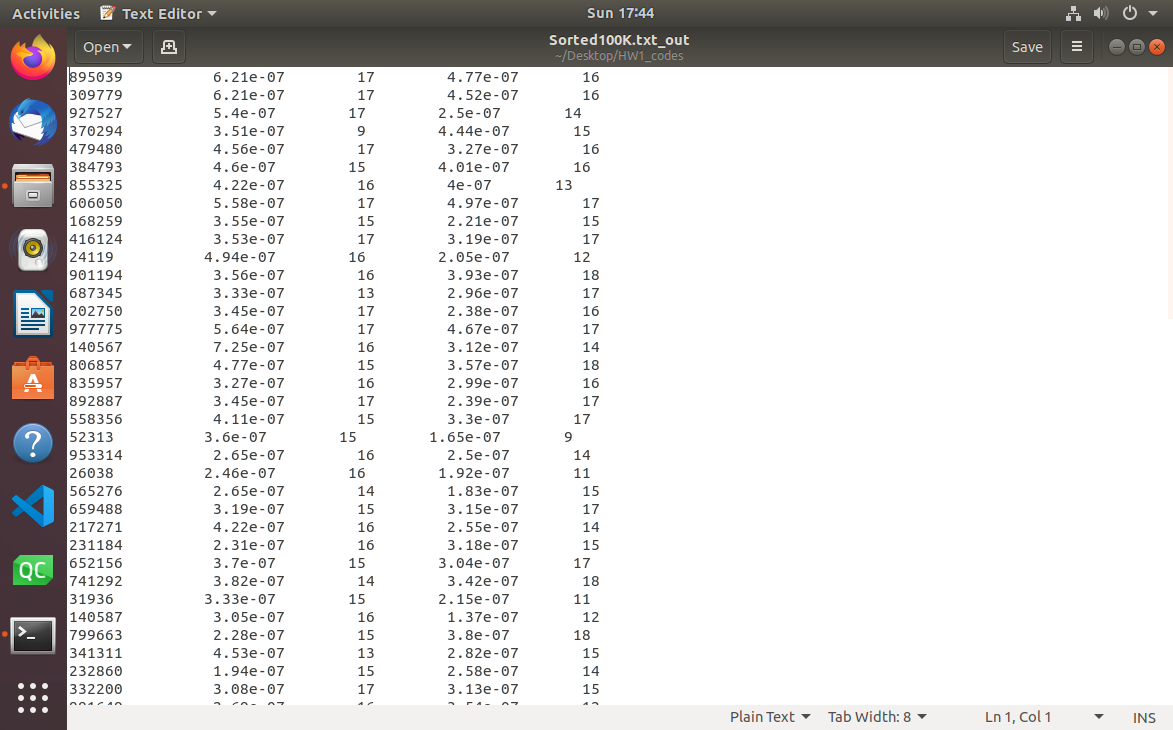


Fig3. Output file (Sorted100K.txt\_out)

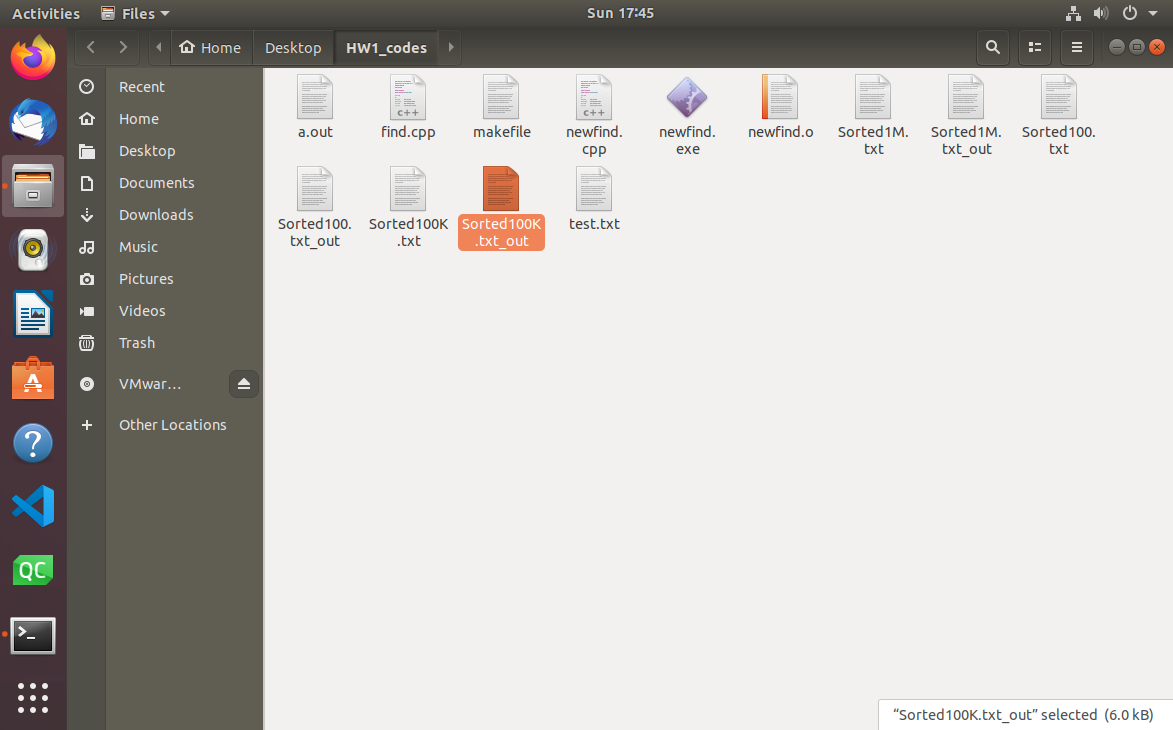


Fig4. All files I got in the HW1\_codes folder