Nathan Bouquet

Paper Assignments Lab 2

Assignment 1:

The goal is to store information with a key range of 40 numbers {0, 1, ..., 39} in a hash table of size 16.
The two hash functions are: h1(k) = k%16 and h2(k) = k/16 (integer division).
For h1(k), we can see :

Number	k%16
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
16	1
17	1
18	2
19	3
20	4
21	5
22	6
23	7
24	8
25	9
26	10
27	11
28	12

29	13
30	14
31	15
32	2
33	1
34	2
35	3
36	4
37	5
38	6
39	7

For h2(k), we can see :

Number	k/16
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0
16	1
17	1
18	1
19	1
20	1
21	1
22	1

23	1
24	1
25	1
26	1
27	1
28	1
29	1
30	1
31	1
32	2
33	2
34	2
35	2
36	2
37	2
38	2
39	2

We know that the hashtable is of size 16.
We can see that the function h1(k) goes to 16 key values from 0 to 15.
Load factor : 40/16 = 2,5
We can see that the function h2(k) goes to only 3 key values, which are 0, 1 and 2.
Load factor : 40/3 = 13,33
So the function h1(k) would perform better for this problem, as it has 5 times more collisions than h2(k) and a much lower load factor.

Assignment 2:

h(“Wojtek”) = 4
h(“peter”) = 4 Not possible
Various techniques that utilize hash values to improve the efficiency of searching a linked list are

Assignment 3:

Big-O (O) notation is a notation to describe the upper bound of the running time of an algorithm.
The running time of algorithm A being at least O(n**2) tells me that the upper bound for the running time is at least O(n**2), which is smaller than O(n!). That information is not useful as the running time of an algorithm with at least O(n**2) is the same as O(n**3) and O(n**4). It doesn’t tell me anything about the lower bound, as the big O notation describes the upper bound.

Therefore, the running time of algorithm A being at least O(n**2) doesn't give me useful information about its running time.

Assignment 4:

1. Is 2**{n + 1} = O(2**n)?
   Knowing that we need to use the highest polynomial degree of the class.
   2**{n + 1} is a degree of 1. So 2**{n + 1} is O(n).
   Furthermore : O(2n) doesn't equal O(n). So 2**{n + 1} doesn't equal O(2**n).
   This statement is false.

2. Is 2**2n=O(2n)?
   2**2n is O(2**n)
   The growth rate of 2**2n is faster than that of 2**n, and 2**2n does not have the same asymptotic behavior as O(2**n).
   This statement is false.

Assignment 5:

Algorithm loop1: this algorithm adds a constant i to a variable s till n. The operations increase linearly according to the input n.
Therefore, the worst case scenario for this algorithm is O(n).

Algorithm loop2: this algorithm adds to a variable p ,equal to 1, p * i till 2n. n being the input. 2n is O(n). So, the worst case scenario for this algorithm is O(n).

Algorithm loop3: this algorithm is nearly the same as loop2. However, the variable p increases till n**2. And n**2 is O(n**2).
So, the worst case scenario for this algorithm is O(n**2).

Algorithm loop4: this algorithm is a nested for loop. The first one goes from i (equal to 1) to 2n. n being the input. And the second one goes from j (equal to 1) to i.
This algorithm being a nested for loop we need to use multiplication to work out the notation.
First for loop : 2n is O(n).
Second for loop : O(i)
So: O(n) * O(i) = O(n*n) = O(n*i).
So, the worst case scenario for this algorithm is O(n**2).

Algorithm loop5: this algorithm is a nested for loop. The first one goes from i (equal to 1) to n**2. n being the input. And the second one goes from j (equal to 1) to i.
First for loop : n**2 is O(n2).
Second for loop : O(i)
So: O(n**2) * O(i) = O(2n*i) = O(in2).
So, the worst case scenario for this algorithm is O(n**4).
And the total of operations is : ((n**2+1)*(n**2))/2