Midterm Assignment

**Due – Mar 19 at 6:00 PM, at the class beginning**

All work must your own. Write your answers on these pages and show your work. If you feel that a question is not fully specified, state any assumptions you need to make in order to solve the problem.

Upload your answers on the blackboard to MidTerm Assignment in the CS566\_A2 course site under assignments.

**No extensions or late submissions for anything** other than major emergency

Write your name and ID on this page

Name Kevin Lopez Sepulveda

Student ID U59702827

# Problem 1 [ 25 pts]

**Implement the following algorithm (it means create the code in any programming language that you are familiar with):**

**Let C(n,k) be the number of combinations of *n* items taken *k* at a time.**

**C(*n*,*k*) = 1 if *n* = *k* or *k* = 0**

**else**

**C(*n*,*k*) = C(*n* - 1,*k* - 1) + C(*n* - 1,*k*)**

**Give a tight asymptotic bound on the worst-case running time by solving the recurrence.**

**Verify your answer by running the code for n=6, 10, 16 and k=n/2.**

**Is a recursive approach the best one?**

The algorithm for calculating the number of combinations C(n, k) using recursion can be analyzed for its worst-case running time. The recurrence relation for the recursive function C(n, k) can be expressed as T(n, k) = T(n - 1, k - 1) + T(n - 1, k) + O(1). The base cases happen when n equals k or k equals 0, which gives us T(n, k) = O(1). To solve this recurrence, we can think of it as a binary tree where each node represents a call to C(n, k). Each node has two children: one for C(n - 1, k - 1) and one for C(n - 1, k). The height of the tree is n, since at each level, n decreases by 1 until it hits 0 or k. The number of leaves, representing the base cases, can be counted as O(2^n) because the number of calls doubles at each level until the maximum depth. So, the tight asymptotic bound on the worst-case running time of the recursive algorithm is T(n) = O(2^n).

To check this, I ran the recursive function in Python for n equals 6, 10, and 16 with k being n divided by 2. The results for C(n, k) are C(6, 3) equals 20, C(10, 5) equals 252, and C(16, 8) equals 12870. While the recursive approach is straightforward, it’s not the most efficient because of its exponential time complexity of O(2^n). This happens mainly because it recalculates the same subproblems multiple times. Better approaches would be using dynamic programming, which lets us store previously computed values in a table, bringing down the time complexity to O(n times k) while using O(n times k) space. Memoization is another option that’s similar to dynamic programming but focuses on caching the results of function calls. So, in the end, while the recursive method is easy to understand and implement, it’s not the best choice for larger values of n and k. Using dynamic programming would be a more efficient way to calculate combinations.

Problem 2 [15 pts]

Assume the complete binary tree numbering scheme used by Heapsort and apply the Heapsort algorithm to the following key sequence (3,25,9, 35,10,13,1,7,46,2,51). The first element index is equal 0.

1. **What value is in location 5 of the initial HEAP?**

**b) After a single deletion (of the element at the heap root) andtree restructuring, what value is in location 5 of the new HEAP?**

To solve the problem, we first apply the Heapsort algorithm to the key sequence 3, 25, 9, 35, 10, 13, 1, 7, 46, 2, 51 and construct a max heap. The sequence is transformed into a complete binary tree where the maximum value is at the root. After building the max heap, the value at index 5 is 9. Next, we perform a single deletion of the root, which contains the maximum value. We replace the root with the last element in the heap and then restructure the heap to maintain the max heap property. After the restructuring, the value at index 5 remains 9. Therefore, the answer to part a is that the value in location 5 of the initial heap is 9, and the answer to part b is that after the deletion and tree restructuring, the value in location 5 of the new heap is also 9.

**Problem 3 [15 pts]**

**Assume that we are given n pairs of items as input, where the first item is a**

**number and the second item is one of three colors (red, blue, or yellow). Further**

**assume that the items are sorted by number. Give an O(n) algorithm to sort the items by color (all reds before all blues before all yellows) such that the numbers**

**for identical colors stay sorted.**

**For example: (1,blue), (3,red), (4,blue), (6,yellow), (9,red) should become (3,red),**

**(9,red), (1,blue), (4,blue), (6,yellow).**

To solve this problem in linear time, we can use a stable counting sort approach since there are only three possible colors: red, blue, and yellow. The idea is to go through the list once and separate the items into three different lists based on their colors. Since the input list is already sorted by number, we just need to make sure we maintain that order within each color group. First, we create three empty lists for red, blue, and yellow. Then, we go through the input list and add each item to the correct list based on its color. After that, we combine all three lists, putting reds first, blues second, and yellows last. Since we are just going through the list a couple of times and doing simple operations, the whole process runs in linear time. This method makes sure that numbers within each color stay in the same order as they were originally given.

**Problem 4 [25pts] Insert the keys <13, 19, 35, 71, 31, 6, 23, 4,98,101> into hash table of size m=13 using linear hashing. Here, *h(k, i) =((k mod m) + i) mod m, i=0,1,2,….***

How many times you increment *i* to resolve collisions?

Does double hashing with h2(k,i)= (k mod 7+i) help to minimize number of increments ?

To insert the given keys into a hash table of size 13 using linear probing, we use the hash function h(k, i) = ((k mod 13) + i) mod 13, where i increases to fix collisions. First, 13 goes to index 0, no problem. Then 19 lands at index 6, 35 at index 9, and 31 at index 5, all without any issues. When inserting 71 at index 6, there is a collision, so i goes up by 1 and it gets placed at index 7. Then 6 tries to go to index 6, but it is already taken. i increases to 1, but index 7 is also full. i goes up again to 2, and 6 finally lands at index 8. Most of the other insertions go fine until 98, which hashes to index 7 and collides. i has to increase four times before finding an open spot at index 11. Finally, 101 hashes to index 10, but that is full, so it takes two increments before it can go to index 12. In total, there were nine increments to fix collisions. Using double hashing with h2(k, i) = (k mod 7 + i) mod 13 would probably help reduce the number of increments because instead of just moving one step at a time, it jumps by different amounts, which spreads things out better and avoids a bunch of collisions in the same area.

Problem 5 [20 pts]

**Describe an efficient algorithm that, given n random integers in the range of 1 to k, preprocess the input and then answers any query about how many of the n integers fall into the range [a..b] in**

O(1) time.

To efficiently answer range queries in constant time, we can preprocess the input using a frequency array and a prefix sum array. First, we create a frequency array of size k+1, initialized to zero, and count how many times each number appears in the input list. Once we have the frequency of each number, we build a prefix sum array where each index stores the sum of all previous counts up to that point. This allows us to quickly compute how many numbers fall within any given range. When answering a query for the range [a..b], we simply compute prefix[b] - prefix[a-1], which gives us the count instantly. The preprocessing step runs in O(n + k) time because we iterate through the list once to count occurrences and then make a single pass to compute the prefix sum. After that, each query runs in O(1) time since it only requires a simple subtraction. This approach ensures that once the input is processed, any range query can be answered in constant time, making it highly efficient.