HW1

2024-10379

June 16 2025

1 Chapter 1

1. 1-1) Show that $a + b$ can be less than $min(a, b)$.

   Given $a = -5, b = -3$, $min(a, b) = -5$ and $a + b = -8$, where $-8 < -5$

   therefore, $a + b$ can be less than $min(a, b)$.

2. 1-2) Show that $a * b$ can be less than $min(a, b)$.

   Given $a = 1/2, b = 1/4$, $min(a, b) = 1/4$ and $a * b = 1/8$, where $1/8 < 1/4$

   therefore, $a * b$ can be less than $min(a, b)$.

3. 1-8) Prove the correctness of the following algorithm for evaluating a polynomial.

   $P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots a_1 x + a_0$

   ```
   function horner(A, x)
     p = A_n
     for i from n − 1 to 0
       p = p · x + A_i
     return p
   ```

   Base Case ($n = 0$):

LHS:	RHS:
horner(A, x)	a_0
a_0	

   Inductive Case $n = n_0$:

   Assume horner$(A, x) = a_{n_0} x^{n_0} + a_{n_0-1} x^{n_0-1} + \cdots + a_1 x + a_0$

   Therefore horner(A, x) should equal $a_{n_0+1} x^{n_0+1} + a_{n_0} x^{n_0} + \cdots + a_1 x + a_0$ when $n = n_0 + 1$

LHS:	RHS:
horner(A, x) where $n = n_0 + 1$	$a_{n_0+1}x^{n_0+1} + a_{n_0}x^{n_0} + \cdots + a_1x + a_0$
$(a_{n_0+1}x^{n_0} + a_{n_0}x^{n_0-1} + \cdots + a_2x + a_1) * x + a_0$	
$\boxed{a_{n_0+1}x^{n_0+1} + a_{n_0}x^{n_0} + \cdots + a_2x^2 + a_1x + a_0}$	$\boxed{a_{n_0+1}x^{n_0+1} + a_{n_0}x^{n_0} + \cdots + a_2x^2 + a_1x + a_0}$

4. 1-10) Prove that $\sum_{i=1}^{n} i = n(n+1)/2$ for $n \geq 0$, by induction.

   Base Case ($n = 1$):

LHS:	RHS:
$\sum_{i=1}^{n} i$	$\frac{n(n+1)}{2}$
$\sum_{i=1}^{1} i$	$\frac{1(1+1)}{2}$
1	1

   Inductive Case $n = n_0$:

   Assume $\sum_{i=1}^{n_0} i = \frac{n_0(n_0+1)}{2}$

   Therefore $\sum_{i=1}^{n_0+1} i$ should equal $\frac{(n_0+1)(n_0+2)}{2}$

LHS:	RHS:
$\sum_{i=1}^{n_0+1} i$	$\frac{(n_0+1)(n_0+2)}{2}$
$\sum_{i=1}^{n_0} i + (n_0 + 1)$	
$\frac{n_0(n_0+1)}{2} + (n_0 + 1)$	
$\frac{n_0(n_0+1)}{2} + \frac{2(n_0+1)}{2}$	
$\frac{n_0(n_0+1)+2(n_0+1)}{2}$	
$\boxed{\frac{(n_0+1)(n_0+2)}{2}}$	$\boxed{\frac{(n_0+1)(n_0+2)}{2}}$

5. 1-11) Prove that $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$ for $n \geq 0$, by induction.

   Base Case ($n = 1$):

LHS:	RHS:
$\sum_{i=1}^{n} i^2$	$\frac{n(n+1)(2n+1)}{6}$
$\sum_{i=1}^{1} i^2$	$\frac{1(1+1)(2(1)+1)}{6}$
1	1

Inductive Case $n = n_0$:

Assume $\sum_{i=1}^{n_0} i^2 = \frac{n_0(n_0+1)(2n_0+1)}{6}$

Therefore $\sum_{i=1}^{n_0+1} i^2$ should equal $\frac{(n_0+1)(n_0+2)(2n_0+3)}{6}$

LHS:	RHS:
$\sum_{i=1}^{n_0+1} i^2$	$\frac{(n_0+1)(n_0+2)(2n_0+3)}{6}$
$\sum_{i=1}^{n_0} i^2 + (n_0+1)^2$	
$\frac{n_0(n_0+1)(2n_0+1)}{6} + (n_0+1)^2$	
$\frac{n_0(n_0+1)(2n_0+1)}{6} + \frac{6(n_0+1)^2}{6}$	
$\frac{(6(n_0+1)+(2n_0+1)(n_0))(n_0+1)}{6}$	
$\frac{((6n_0+6)+(2n_0^2+1n_0))(n_0+1)}{6}$	
$\frac{(2n_0^2+7n_0+6)(n_0+1)}{6}$	
$\boxed{\frac{(2n_0+3)(n_0+2)(n_0+1)}{6}}$	$\boxed{\frac{(n_0+1)(n_0+2)(2n_0+3)}{6}}$

2 Chapter 2

1. 2-5) Suppose the following algorithm is used to evaluate the polynomial $p(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$

$p := a_o$;
$xpower := 1$;
for $i := 1$ to n do
  $xpower := x * xpower$;
  $p := p + a_i * xpower$
end

(a) How many multiplications are done in the worst-case? How many additions?

In the worst case, multiplications will be done a total of 2n times while additions happen n times.

(b) How many multiplications are done on the average?

On average, multiplications are done 2n times.

(c) Can you improve this algorithm?

Horner's method, found in 1-8, is a more efficient algorithm for evaluating a polynomial where only n multiplications are performed instead of $n - 1$.

2. 2-13) Prove that if $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then $f_1(n) + f_2(n) = O(g_1(n) + g_2(n))$.

   By the definition of Big-O notation:

   there exists $c_1 > 0$, n_1 such that $f_1(n) \leq c_1 g_1(n)$ for all $n \geq n_1$,

   and $c_2 > 0$, n_2 such that $f_2(n) \leq c_2 g_2(n)$ for all $n \geq n_2$.

   Let $n_0 = \max(n_1, n_2)$. Then for all $n \geq n_0$, we have:

   $$f_1(n) + f_2(n) \leq c_1 g_1(n) + c_2 g_2(n).$$

   Let $c = \max(c_1, c_2)$. Then:

   $$f_1(n) + f_2(n) \leq c(g_1(n) + g_2(n)).$$

   By definition of Big-O:

   $$f_1(n) + f_2(n) = O(g_1(n) + g_2(n)).$$

3. 2-32) Prove that: $1^2 - 2^2 + 3^2 - 4^2 + \cdots + (-1)^{k-1}k^2 = \frac{(-1)^{k-1}k(k+1)}{2}$

   Base Case ($k = 1$):

LHS:	RHS:
1^2	$\frac{(-1)^{1-1}1(1+1)}{2}$
1	$\frac{(-1)^0(2)}{2}$
1	1

Inductive Case $k = k_0$:

Assume $(1^2 - 2^2 + 3^2 - 4^2 + \cdots + (-1)^{k_0-1}k_0^2 = \frac{(-1)^{k_0-1}k_0(k_0+1)}{2}$

Therefore $(1^2 - 2^2 + 3^2 - 4^2 + \cdots + (-1)^{k_0}(k_0+1)^2$

should equal $\frac{(-1)^{k_0}(k_0+1)(k_0+2)}{2}$

LHS:	RHS:
$\frac{(-1)^{k_0-1}k_0(k_0+1)}{2} + (-1)^{k_0}(k_0+1)^2$	$\frac{(-1)^{k_0}(k_0+1)(k_0+2)}{2}$
$\frac{(-1)^{k_0-1}k_0(k_0+1)}{2} + \frac{2(-1)^{k_0}(k_0+1)^2}{2}$	
$\frac{(-1)^{k_0-1}k_0(k_0+1)+2(-1)^{k_0}(k_0+1)^2}{2}$	
$\frac{(k_0+1)((-1)^{k_0-1}k_0+2(-1)^{k_0}(k_0+1))}{2}$	
$\frac{(k_0+1)(-1)^{k_0}(\frac{k_0}{-1}+2(k_0+1))}{2}$	
$\frac{(k_0+1)(-1)^{k_0}(-k_0+2k_0+2)}{2}$	
$\boxed{\frac{(k_0+1)(-1)^{k_0}(k_0+2)}{2}}$	$\boxed{\frac{(-1)^{k_0}(k_0+1)(k_0+2)}{2}}$

4. 2-42) In one of my research papers I give a comparison-based sorting algorithm that runs in $O(n\log(\sqrt{n}))$. Given the existence of an $\Omega(n\log n)$ lower bound for sorting, how can this be possible?

   $n\log(\sqrt{n}) = n\log(n^{1/2}) = \frac{1}{2}n\log(n)$

   $\frac{1}{2}n\log(n)$ is in the same order as $n\log(n)$, since $\frac{1}{2}$ is a constant and negligible, so the situation is possible, as the latter does not dominate the former.

5. 2-46) You have a 100-story building and a couple of marbles. You must identify the lowest floor for which a marble will break if you drop it from

this floor. How fast can you find this floor if you are given an infinite supply of marbles? What if you have only two marbles?

With infinite marbles, we drop at the midpoint of the possible floors, that is, the 50^{th} floor. Whether the marble breaks determines whether the 50^{th} floor becomes the upper or lower bound for the next test, where it becomes the upper bound if it breaks, and the lower bound otherwise. We repeat the process, choosing the midpoint of the possible floors every time until we determine the lowest floor at which the marble breaks. This results in a worst-case of 7 marbles dropped. Generally, the worst case would follow $O(\log_2 n)$

With one marble, we must drop it linearly starting from floor 1 and going through all floors to floor 100. With two marbles, we can afford to lose one marble; therefore, we drop the first marble at a certain interval of floors, essentially segmenting the building, such that the second marble needs only to go through all floors within a segment. Specifically, the interval at which the first marble is dropped should decrease by one for every segment, such that the worst case for every segment would require the same number of drops and would be equal to the first floor on which the marble at. Mathematically, the least drops happen when n is smallest, where,

$\sum_{i=0}^{n}(n-i) >= 100$

which is $n = 14$.

3 Chapter 3

1. 3-5) Find the overhead fraction (the ratio of data space over total space) for each of the following binary tree implementations on n nodes:

   (a) All nodes store data, two child pointers, and a parent pointer. The data field requires four bytes, and each pointer requires four bytes.

   Each node takes up 4 bytes for data, 8 bytes for its child pointers, and 4 bytes for its parent pointer, totaling 16 bytes.

   $\frac{DataSpace}{TotalSpace} = \frac{4n}{16n} = \boxed{\frac{1}{4}}$

   (b) Only leaf nodes store data; internal nodes store two child pointers. The data field requires four bytes, and each pointer requires four bytes.

   Each leaf node takes up 4 bytes for data and 6 bytes for pointers, while each internal node takes up 6 bytes for pointers, meaning total space is $10 * n_{leaf} + 6 * n_{internal}$ while data space is just $4 * n_{leaf}$.

$$\frac{DataSpace}{TotalSpace} = \boxed{\frac{4n_{leaf}}{10n_{leaf} + 6n_{interval}}}$$

2. 3-12) Suppose you are given an input set S of n numbers, and a black box that, if given any sequence of real numbers and an integer k, instantly and correctly answers whether there is a subset of the input sequence whose sum is exactly k. Show how to use the black box $O(n)$ times to find a subset of S that adds up to k.

```
S = {s_1, s_2, s_3, ··· , s_n}
function blackBox(S, k) → bool # O(1)
  if there exists S_0 ⊆ S where sum(S_0) = k
    return true
  else
    return false
if blackBox(S, k) # O(n)
  for S_i in S
    if blackBox(S − S_i, k)
      S = S − S_i
  return S
else
  return "No such subset in S"
```

3. 3-18) What method would you use to look up a word in a dictionary?

   I would match the first letter in the word to an entry in the dictionary then continue. If possible it would be best to go to the middle of a section that satisfies the current letters that I have matched.

4. 3-19) Imagine you have a closet full of shirts. What can you do to organize your shirts for easy retrieval?

   Shirts could be organized by color or type. Since there will usually multiple of either attribute it's best to sort by all following a heirarchy, such that we would sort by type first, then by color within each type, and can further sort by hue for each color.

5. 3-29) Give an algorithm for finding an ordered word pair (e.g., "New York") occurring with the greatest frequency in a given webpage. Which data structures would you use? Optimize both time and space.

   I would first create a dictionary and assign an integer to each word present in the document. After that we parse the document, converting each word pair into int pairs, which should take $O(1)$, constructing a weight adjacency graph where the x axis is the first word, the y axis is the second word, and the weight is the frequency. We simply use the generated int pair as coordinates to increase the weight of a certain node, which also takes $O(1)$.

4 Ex. 1

1. Give the theoretical worst-case analysis for the time complexities of each strategy.

   linGuess: $O(N)$

   randGuess: $O(\log(N))$

   binGuess: $O(\log(N))$

2. Give the theoretical average-case analysis for the time complexities of each strategy.

   linGuess: $O(N)$

   randGuess: $O(\log N)$

   binGuess: $O(\log N)$

3. If you want to minimize average steps, what is the best strategy and why is it objectively optimal? Note: the best strategy may be one of the strategies given, or you may need to make your own.

   The best startegy is binary guess where the worst case will take n steps where $n^2 - 1 \geq N$. This is because we always eliminate the maximum amount of possibilities with equal probability.

4. How does the optimal strategy change if instead of saying higher or lower, each guess tells you is you are within $\pm\frac{N}{k}$ of the target? For simplicity, assume k is an integer greater than 1.

   With this change, the binary guess is impossible as the bounds remain centered on our initial guess which makes it impossible to guess a number other than n/2.

5 Ex. 2

1. Explain your choice of hashing function.

   The hashing function I chose to use converts a character into an integer according to CP1252. After that it multiplies the integer by 256 to the power of the character's position in the hashed string. The sum of all integers is the hash

2. In the book, the hashes are constructed to be pseudo-random, while Python dictionary hashes are not random, both for different reasons. Is randomness a necessary property of Rabin-Karp hashes?

   Randomness is not a necessary property of the Rabin-Karp algorithm. What is necessary is efficiency timewise as well as to reduce collisions as

much as possible, so as to reduce false-positives and wasted computations to verify.

3. The earliest implementations of Rabin-Karp used the Rabin fingerprint for hashing. What is this fingerprint, and what are its advantages?

   Rabin fingerprint is a hash that turns each character into an integer, multiplies by the number of unique said integers to a power based on the characters position in the string, sums all these processed characters and then divides by some prime number. Hash collisions are rare because of the large base, it allows constant-time updates, and is optimized for substring matching.

4. How would you modify your function to search multiple patterns?

   I would turn all patterns into their hashes, grouped by the length of each string, then it would roll through with windows of each length for which a pattern exists. For patterns of the same length, I would compare the hash within the same roll.