Please enter your name and uID below.

Name: Devin White

uID: u0775759

Collaborators, if any, and how you collaborated:

Submission notes

- Due at 11:59 pm on Thursday, October 20.
- Solutions must be typeset using one of the template files. For each problem, your answer must fit in the space provided (e.g. not spill onto the next page) *without* space-saving tricks like font/margin/line spacing changes.
- Upload a PDF version of your completed problem set to Gradescope.
- Teaching staff reserve the right to request original source/tex files during the grading process, so please retain these until an assignment has been returned.
- Please remember that for this problem set, you are allowed to collaborate in detail with your peers, as long as you cite them. However, you must write up your own solution, alone, from memory. If you do collaborate with other students in this way, you must identify the students and describe the nature of the collaboration. You are not allowed to create a group solution, and all work that you hand in must be written in your own words. Do not base your solution on any other written solution, regardless of the source.

1. (Basic arithmetic expression)
   (a) if n = 1 then returns 1. This is the base case
   (b) if the last expression used was addition the first term will have BAE(i) ones, while the second term, n-i contains BAE($n - i$) ones. This applies up to n/2 since the first term at most can only ever be equal to n/2, and is usually less than this.(eg. 4 is equivalent to 1 + 3 where 1 is (i) and 3 is (n-i) and so on
   (c) if the last expression used was multiplication, the first term will again have BAE(i) ones, while the second term will have BAE($\frac{n}{i}$) ones, IF n is divisible by i. this applies while i is less than $\sqrt{n}$ for example, if n is 4, BAE(4) = BAE(2) * BAE(2)

$$\text{BAE(n)} = \begin{cases} 1 & n = 1 \\ 0(\text{or } \infty?) & n < 1 \\ min \begin{cases} min \begin{cases} BAE(i) + BAE(n-i) & \text{if } 1 \le i \le \frac{n}{2} \\ n \end{cases} \\ min \begin{cases} BAE(i) + BAE(\frac{n}{i}) & \text{if } 1 \le i \le \sqrt{n} \\ n \end{cases} \end{cases} & otherwise \end{cases}$$

The recursive subproblems being solved is thus: How many 1's are needed in order to solve each smaller value? E.G if n = 1, it requires 1, the num of 1's = n up to 4. With these smaller problems, you can solve other larger subproblems such as the number of 1's to get 5 being BAE(3) + BAE(2)) or 5, the minimum number of 1's in 6 being 5((1+1+1)*(1+1)) which is equivalent to BAE(2)+BAE(3)) the number of 1's to get 7 being BAE(6) + BAE(1)((1+1)*(1+1+1)+ 1) which can be further decomposed into BAE(3) + BAE(2) + BAE(1)((1+1+1) * (1+1) + 1)) going further, BAE(20)=BAE(10)+BAE(2) = BAE(5)+BAE(4) and so on and so forth. The memoization structure for this algorithm can be accomplished with a simple 1-D array of size n. As each subsequent value can be obtained from some previous value, where BAE[i] depends on the previous entries BAE[j] where $j < i$ (e.g BAE(2) is just BAE(1)+BAE(1)). This means that the array will be filled increasing order, from the initial index to n. We also know that the initial index(base case) is just 1, so we can fill that immediately. Three loops in total are required to fill the memoization structure, the initial $i \to 2ton$ loop, and 2 seperate loops nested within that loop to account for the recursive if statements $j \to 1$ to $i/2$ and $j \to 2$ to $\le \sqrt{i}$. This Gives us $O(n) * (O(n) + O(n))$ or $O(n * 2n)$ which removing the constant means the algorithm runs in approximately O(n^2) time. **The PseudoCode follows:**

Algorithm 1: Basic Arithmetic Expression

```
input  : a number n
output: number of 1's to form n
BAE[1]=1;
for $i \leftarrow 2$ to $n$ do
    $BAE[i] = i$;
    for $j \leftarrow 1$ to $\frac{i}{2}$ do
        if (BAE[I] > BAE[j] + BAE[i+j]) then ;
        $BAE[i] = BAE[j] + BAE[i-j]$
    for $j \leftarrow 2$ to $\sqrt{i}$ do
        if i % j == 0 then ;
        if (BAE[j] + BAE[$\frac{i}{j}$] ¡ BAE[i]) then ;
        $BAE[i] = BAE[j] + BAE[\frac{i}{j}]$
```

2. (Square peg in a round hole)

The first thing that needs to be done for this algorithm is to convert the square houses into a circular house of corresponding size such that the original square house with a side size of s fits **inside** of the circular equivalent. Given the area of a square can be expressed as s^2 and the area of a circle can be expressed as πr^2. A square fits into a given circle of radius r if and only if $\frac{s}{\sqrt{2}} < r$ Since the diagonal "diameter" of a square can be expressed as $s * \sqrt(2)$ and the radius is 1/2 diameter, the radius of the circle is equivalent to $\frac{s*\sqrt{2}}{2}$ for example, a square house of size 4 will fit into a circle with a radius of 2.828425 ($\frac{4*\sqrt{2}}{2} = 2.828425$) After converting the square houses to fit inside their circular equivalent, we can combine the two lists of houses into a single list of houses which we will call H, of size H[1...m+k], by whatever means. such as H[1..m + k] = Append(M, S). After the houses are combined into a single list of **circular** houses H, we are now left with 2 lists, one consisting of plot sizes N[i...n], and one of houses H[1..m+k]. **The algorithm works as thus: *1.* sort the plots and houses from smallest to largest *2.* look through the list of houses one by one in reverse(descending) order(going from largest to smallest)Finding the largest possible house that can fit in a given plot, thus leaving smaller plots open for smaller houses *3.* upon finding a house that fits within a plot, place the house within that plot(remember, going from largest plots and houses to smallest plots and houses) *4.* move to the next plot(by decrementing the plot index) and repeat the above until all houses are placed in plots, all plots are filled, or there are no more houses that fit within a plot(Note, this can also be done by sorting from largest to smallest and going through the list in normal order it doesn't matter so long as the largest possible house that can fit in any given plot is placed in that plot)**
proving correctness Assume The lists of houses and plots are sorted in descending order as this will not affect the correctness of our proof. In addition, assume that in any perfect(all houses placed or all plots filled) solution the most possible houses that can be placed in plots is the number of houses or the number of plots, whichever is smaller Also Assume, for the sake of contradiction, that the proposed greedy algorithm is not the optimal solution. This means that for some input, if our greedy algorithm matches p houses to plots, then there is some different algorithm that matches a larger number, $q > p$ houses to plots. We can represent the output of a particular algorithm by listing the plot indices it chooses for each house, or -1 if it did not match the house. So, these two outputs can be represented with two sequences: **greedy:** hg1, . . . , gj , . . . , gk+mi **optimal** ho1, . . . , oj , . . . , ok+mi Where g has a total of p non-negative plot indices in the whole list and o has a total of q(this is to say, g is the greedy solution, and o is the hypothetical optimal solution). As these lists are sorted in descending order, each subsequent array element is greater than or equal to the previous array element, that is to say N[i] $\geq$ N[i+1] $\geq$ N[i+2] and H[i] $\geq$ H[i+1] $\geq$ H[i+2] it is assumed that the optimal solution follows the same logic. Since this hypothetical optimal solution is NOT the greedy solution, it can be assumed that for some index i, gi $\neq$ oi, therefore, since gi finds the largest possible house that can be placed in a plot, it **must** be the case oi < gi when oi $\neq$ gi, furthermore, since the plots are whole integers, oi $\leq$ gi - 1 or put another way, gi > oi+1. This means that starting from the point where the plots differ, the smaller house in the optimal solution will be placed within a larger plot with "wasted space". As gi picks the largest house which will fit, gi can always be swapped with Oi without affecting oi. By swapping Oj in the optimal solution with gj in the greedy solution, we now have the largest house that can fit in the plot at Nj, meaning that there are more possible houses to choose from for fitting in the smaller plots. further, if oi does not fit in a plot(-1) then Gi also does not fit in that plot, and gi and oi can be swapped without affecting the running time of oi.

These two cases leads to a contradiction, as if Oi as THE optimal solution, then swapping elements in O with elements in g should negatively affect the running time,but it doesn't.