1. What is the running time of the below code?

Algorithm 1 Test function

```
Test(a, start, end) :
n = end - start ;
if n <= 1
    return a[n] ;
else
    newEnd = start + $\frac{n}{6}$ ;
    newEnd2 = newEnd + $\frac{2 \cdot n}{6}$ ;
    sol1 = Test(a, start, newEnd) ;
    sol2 = Test(a, newEnd + 1, newEnd2) ;
    sol3 = Test(a, newEnd2 + 1, end) ;
    combineSol = combine(a, start, newEnd, end) //$T(n) = O(n)$
    return min([sol1,sol2, sol3, combineSol]) ;
end
```

2. You are given a matrix called buildings that has locations of all the buildings at a university in a two-dimensional coordinate. We would to construct paved paths that connected the buildings to each other. Implement an algorithm to calculate the minimum budget required to finish the constructions.

1. How would you find the minmum amount to construct the paths?
2. Write the pseudocode for the best algorithm you came up with.
3. Implement your answer using any programming language you want to.
4. What is the time complexity of your answer?

3. You are given an adjaccency matrix that has 0s and 1s in it. Implement an algorithm to find the exact number of connected components on the map.

1. How can you find the total number of connected components?

   A connected component is a maximal subgraph such that there exist a walk between any two vertices in that subgraph. We know that if we choose an arbitrary vertex and generate a walk to all vertices that it could walk to via DFS or BFS then that constitutes a connected component. If a vertex has not been visited then, since the connected component is maximal, this vertex must not be in the connected component and we compute the walks again for that connected component. This gives us a way to compute the connected components. we start a counter initialized to 0. We iterate through the vertex set and if we have not seen a particular vertex that means we have a new connected component, compute DFS on that vertex to find and mark all nodes that are visited. Incrementing the counter by one and again iterate through the vertex set computing DFS and then incrementing the counter if we have an unseen vertex.

2. Write the pseudocode for the best algorithm you came up with.

Algorithm 2 Connected components

```
connected_components(G) :
 1: m = G.rows()
 2: n = G.cols()
 3: cc = 0
 4: visited[m][n] = {false} // initialize all vertices to not visited
 5: for(i = 0 ;i < m ;i++) :
 6:     for(j = 0 ;j <n ;j++) :
 7:         if not visited[i][j] :
 8:             DFS(i, j, m, n, visited)
 9:             cc++
10: return cc
DFS(i, j, m, n, visited) :
 1: visited[i][j] = true ;
 2: possible_neighbors = {[0, 1], [0, -1], [1, 0], [-1, 0], [1, 1], [1, -1], [-1, 1], [-1,
    -1]} // neighbors can be to the left,right and diagonals of a possible vertex
 3: for neighbor in possible_neighbors :
 4:     new_i = neighbor[0] + i
 5:     new_j = neighbor[1] + j
 6:         if not out_of_bounds(new_i, new_j, m, n) and not visi-
    ted[new_i][new_j] :
 7:         DFS(new_i, new_j, m, n, visited)
```

3. Implement your answer using any programming language you want to.

4. What is the time complexity of your answer?

   We analyze the pseudocode, suppose that we are given a matrix with no 1's, that is a matrix with 0 connected components. The algorithm will iterate through all entries and never compute DFS. Since the matrix will contain $m \cdot n$ elements this will be $\Theta(m \cdot n)$. Suppose the matrix is all 1's. Then DFS will be called on the leftmost element and continously be called throughout the matrix since the graph is fully connected. Any call will have $\Theta(1)$ work but there will be $\Theta(m \cdot n)$ calls resulting in $\Theta(m \cdot n)$. Once the connected component has been computed, the algorithm will still iterate through the rest of the vertex set, since there exists $m \cdot n$ total elements, this is $\Theta(m \cdot n)$. The total running time for this case is

$\Theta(m \cdot n) + \Theta(m \cdot n) = \Theta(m \cdot n)$. Any other case would be a mixture of these two cases and so the algorithm in general is $\Theta(m \cdot n)$.