2.1

- A* under consistent and admissible heuristic will output the minimum cost path.
- BFS outputs the path that has the lowest number of edges regardless of edge weights
- BFS outputs N0-N5.
- This means A* should also return N0-N5.
- This means edge weight N0-N5 should be lesser than the second lowest cost path (N0-N1-N2-N5).
- Edge weight(N0-N5) < 1 + 2+ 3
- Edge weight(N0-N5) in (0, 6)

Correct option: E

2.2

Given h(N6) = 7
Consistency implies
h(N5) <= c(N5-N6) + h(N6)
h(N5) <= 17

h(N4) <= c(N4-N5) + h(N5)
h(N4) <= 10 + h(N5)
h(N4) <= 27
Given h is non-negative
h(N4) >= 0

h(N4) is in [0, 27]

Does admissibility hold through the interval?
h*(N4) = 30 so admissibility holds throughout the interval [0, 27]

Can consistency hold throughout the interval [0, 27]?
TLDR; yes

Between any two neighboring nodes p and q
h(p) <= c(p-q) + h(q)
h(p) <= 10 + h(q) —- eq1

And also, in the other direction
h(q) <= c(p-q) + h(p)
h(q) <= 10 + h(p) —- eq2

CASE 1: h(N4) in [0, 17]
Lets us say h(N4) = x, where x in [0, 17]
We design a heuristic h such that

- h(N1) = h(N2) = h(N4) = h(N5) = h(S) = x
- h(N3) = h(N6) = 7

- h(T) = 0

Eq1 and eq2 hold for all pairs of neighbors with this designed h.

CASE 2: h(N4) in [17, 27]

Let us say h(N4) = x, where x in [17, 27]

We design a heuristic h such that

- h(N1) = h(N4) = h(S) = x
- h(N2) = h(N5) = 17
- h(N3) = h(N6) = 7
- h(T) = 0

Eq1 and eq2 hold for all pairs of neighbors with this designed h.

Correct option: A

2.3

```
visited[:] = false; //all unvisited nodes
dfs(s, visited, [])

dfs(node, visited, current_path):
   if node is visited:
      return
   visited[node] = true

   if node is target:
      print current_path
      exit

   for nbr in all_neighbors(node): //any neighbor could be expanded based
on tie breaking
      if nbr is not visited:
         current_path.append(nbr)
         dfs(nbr, visited, current_path)
         current_path.pop_back(nbr)
```

Based on the above code and considering there can be various tie breaking strategies. Any path from source to target can be formed by some tie-breaking scheme. For example,

Starting with S

Then unvisited N1 neighbor of S is visited

Then unvisited N2 neighbor of N1 is visited

Then unvisited N4 neighbor of N2 is visited

Finally, T target node is visited for the first time => print the path S-N1-N2-N4-T (option D)

Correct options: B, C, D, E