Name: Christian Whetham
Student Number: 250916490

1 P = babbabbabbababbabb

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
b	a	b	b	a	b	b	a	b	b	a	b	a	b	b	a	b	b
0	0	1	1	2	3	4	5	6	7	8	9	0	1	0	0	1	0

2 Using the normal KMP matching we add components to when the string mismatches and matches on a string. We examine the q value to check the size of the matched string and if it is greater then what we already have then we store the start location of that prefix (q).

3 the function will take the c table, x and y strings and the I and j size values. if the c table for index I,j is 0 then don't print anything. If x[i] and y[j] are equal recursively call with I and j both subtracting 1 from their cost, then print the value of x at index i. otherwise if the c value at i-1,j is greater than I,j-1 then recursively call this function with the I value being i-1. Else recursively call with j now being j-1.

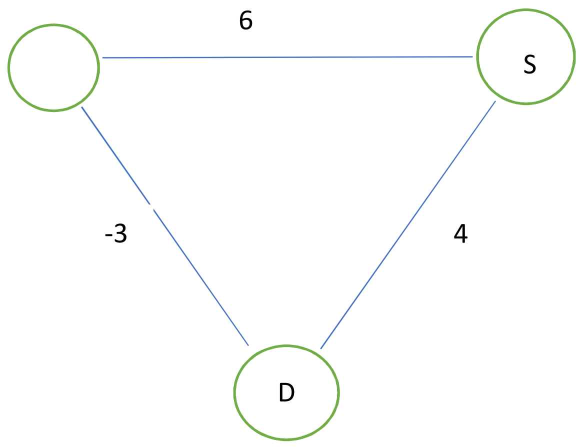
4 We loop through for each item available, each time we check if the current weight plus the new weight exceeded the max weight we can carry. If it doesn't we set the current weight to include the item and add its value to the list. This is correct as the most valuable objects have the smallest weights so we increment from the smallest to largest values until we are full and thus get the greatest total

5 For this question we only need an array that keeps track of the weight (as we have as many items as we want),so this array will be of size w+1 where w is the total weight of that index. From there we set all starting values of the indices to 0 and continue with the normal algorithms double loop. We will set a new max value if the weight of the item is less then the max weight. The max will be the largest value between the array indexes current value and the value of the total weight minus the item plus the value of the item.

6

```
MST Prim(G, w, r)
1 for each u ∈ V [G] do
2    key[u] :=0;
3    π[u] := NIL;
4  key[r] := ∞;
5  Q := V [G];
6  while Q =/= Ø do
7      u := Extract Max( Q);
8      for each v ∈ Adj[u] do
9          if v ∈ Q and w(u, v) > key[v] then
10             π[v] := u;
11             key[v] := w(u, v);
12             update key[v] in Q
```

7



From S to D the algorithm would take the path of weight 4 as it is the shortest of the two, this is incorrect in the end however as the end total of the other path is 3 which is shorter.

8 Yes, it is still correct, the all pair algorithm compares all possible paths through the graph and so unlick Dijkstra's algorithm it will not be fooled by initially large paths. As a result of this it will be able to work around negative weights while going through the search.

9 For each vertex v in V, run BFS on v until v is reached again (or no cycle is found). Record the smallest weight found (if any) and return it.