Question 1: What is the output of the following program?

```
#include <iostream>
using namespace std;

int s(int n)
{
  if (n == 0) return 0;
  return n + s(n-1);
}

int main()
{
  cout << s(10) << endl;
}
```

Question 2: What is the running time of *s(n)*?

(In these questions, use the carat symbol (^) for exponentiation). **Question 3**: What is the output of the following program?

```
#include <iostream>
using namespace std;

string t(string v)
{
  string tmp;
  if (v.size() == 0) return "";
  tmp.push_back(v[0]);
  return t(v.substr(1)) + tmp;
}

int main()
{
  cout << t("Fred") << endl;
}
```

Question 4: What is the running time of *t(v)*, where *n* = **v.size()**?

Question 5: What would be the running time of *t(v)* if **v** were a reference parameter?

Answers to the Clicker Questions

Question 1

- s(10) returns 10 + s(9).
- s(9) returns 9 + s(8).
- s(8) returns 8 + s(7).
- And so on
- s(0) returns 0

So, **s(n)** returns $n + (n-1) + .. + 2 + 1$, which is $n(n+1)/2$. The answer is 55.

Question 2

Answer: It makes n recursive calls, and each call does $O(1)$ work (besides the recursion). So the answer is $O(n)$.

Question 3

- s("Fred") sets **tmp** to "F" and calls s("red"). It returns s("red") + "F".
- s("red") sets **tmp** to "r" and calls s("ed"). It returns s("ed") + "r".
- s("ed") sets **tmp** to "e" and calls s("d"). It returns s("d") + "e".
- s("d") sets **tmp** to "d" and calls s(""). It returns s("") + "d".
- s("") returns "".
- So, s("d") returns "d".
- So, s("ed") returns "de".
- So, s("red") returns "der".
- So, s("Fred") returns "derF".

In other words it reverses the string: "derF".

Question 4

Answer: It makes n recursive calls; However each recursive call creates a substring of size $(n-1)$ and then copies it when calling **s()** on the substring. So the running time of this is $n + (n-1) + (n-2) + ... + 2 + 1$, which is $O(n^2)$.

Question 5

With the reference parameter, a copy of the substring is no longer made. However, making the substring still takes $O(n)$ work, so the running time is still $O(n + (n-1) + (n-2) + ... + 2 + 1)$, which is still $O(n^2)$.