Started on	Sunday, 23 February 2020, 5:58 PM
State	Finished
Completed on	Sunday, 23 February 2020, 6:07 PM
Time taken	9 mins 26 secs
Grade	**10.00** out of 10.00 (**100**%)

Question **1**

Correct

Mark 2.00 out of 2.00

Gene Amdahl, one of the early pioneers in computing, made a simple but insightful observation about the effectiveness of improving the performance of one part of a system. This observation has come to be known as Amdahl's law. The main idea is that when we speed up one part of a system, the effect on the overall system performance depends on both how significant this part was and how much it sped up.

That law says that if a fraction α of the code can be speed up by a factor of k, the total system speedup is

$S = \frac{1}{(1-\alpha)+\alpha/k}$

The marketing department at your company has promised your customers that the next software release will show a 1.92-fold performance improvement. You have been assigned the task of delivering on that promise. You have determined that only 60.0% of the system can be improved. How much (i.e., what value of k) would you need to improve this part to meet the overall performance target?

4.97

Your last answer was interpreted as follows: 4.97

Your answer was 5.0.

Because $S = \frac{1}{(1-\alpha)+\alpha/k}$ then $k = \frac{\alpha S}{(\alpha-1)S+1} = \frac{0.6*1.92}{(0.6-1)1.92+1} = 5$

Correct answer, well done.

A correct answer is 5, which can be typed in as follows: 5

Question **2**

Correct

Mark 2.00 out of 2.00

Suppose you are given the task of improving the performance of a program consisting of three parts. Part A requires 20% of the overall run time, part B requires 30%, and part C requires 50%. You determine that for $1000 you could either speed up part B by a factor of 3.0 or part C by a factor of 1.5. Which choice would maximize performance?

Select one:

- (selected) 1. Speed up part B by a factor of 3.0

✔ Correct - Speeding up part B by 3 gives an overall speedup of 1/(.2+ .3/3+ .5) = 1.25. Speeding up part C by 1.5 gives an overall speedup of 1/(.2+ .3+ .5/1.5) = 1.2.

- 2. Speed up part C by a factor of 1.5

Your answer is correct.

This problem is a simple application of Amdahl’s law. Speeding up part B by 3 gives an overall speedup of 1/(.2+ .3/3+ .5) = 1.25. Speeding up part C by 1.5 gives an overall speedup of 1/(.2+ .3+ .5/1.5) = 1.2.

So the best strategy is to optimize part B.

The correct answer is: Speed up part B by a factor of 3.0

Question **3**

Correct

Mark 2.00 out of 2.00

The following problem illustrates the way memory aliasing can cause unexpected program behavior.

Consider the following function to swap two values:

```
void swap(int *xp, int *yp)
{
  *xp = *xp + *yp;
  *yp = *xp - *yp;
  *xp = *xp - *yp;
}
```

If this procedure is called with **xp** equal to **yp** (i.e. both **xp** and **yp** point to the same integer) what effect will it have compared to the situation where **xp** and **yp** point to different integers?

Select one:

- a. It is not possible for **xp** and **yp** to have the same value.
- b. The value will always be the original value in the integer pointed to by **xp**.
- (selected) c. The value will always be zero.

✔

- d. The value will always be the original value in the integer pointed to by **yp**.
- e. It will be the same - doesn't matter.

Your answer is correct.

This question is addressing the case where **xp** and **yp** both point to the same location.

In that case, the operations would be the same as the following for a value **v:**

v = v + v; // e.g. 2*v
v = v - v; // e.g. now v is zero
v = v - v; // still zero

This example illustrates that our intuition about program behavior can often be wrong. We naturally think of the case where **xp** and **yp** are distinct but overlook the possibility that they might be equal. Bugs often arise due to conditions the programmer does not anticipate.

The correct answer is: The value will always be zero.

Question **4**

Correct

Mark 2.00 out of 2.00

Consider the following functions:

```
int min(int x, int y) { return x < y ? x : y; }
int max(int x, int y) { return x < y ? y : x; }
void incr(int *xp, int v) { *xp += v; }
int square(int x) { return x*x; }
```

Assume x equals 10 and y equals 100. The following code fragment calls these functions.

```
for (i = max(x, y) - 1; i >= min(x, y); incr(&i, -1))
    t += square(i);
```

The function **min** is called 91 ✔ times.

The function **max** is called 1 ✔ times.

The function **incr** is called 90 ✔ times.

The function **square** is called 90 ✔ times.

Question **5**

Correct

Mark 2.00 out of 2.00

Assume you have the following code

```
/* Accumulate in temporary */
void inner4(vec_ptr u, vec_ptr v, data_t *dest)
{
  long int i;
  int length = vec_length(u);
  data_t *udata = get_vec_start(u);
  data_t *vdata = get_vec_start(v);
  data_t sum = (data_t) 0;
  for (i = 0; i < length; i++) {
     sum = sum + udata[i] * vdata[i];
  }
 *dest = sum;
}
```

and you modify the code to the form below.

```
/* Accumulate in temporary */
void inner4(vec_ptr u, vec_ptr v, data_t *dest)
{
  long int i;
  int length = vec_length(u);
  data_t *udata = get_vec_start(u);
  data_t *vdata = get_vec_start(v);
  data_t sum = (data_t) 0;
  for (i = 0; i < length; i += 4) {
     sum = sum + udata[i] * vdata[i]
               + udata[i+1] * vdata[i+1]
               + udata[i+2] * vdata[i+2]
               + udata[i+3] * vdata[i+3];
  }
  for (i = 0; i < length; i++) {
     sum = sum + udata[i] * vdata[i];
  }
 *dest = sum;
}
```

What type of optimizations is being applied?

Select one:

- a. Machine independent optimization
- b. Common subexpression elimination
- c. Inlining
- d. Unrolling and multiple accumulators
- e. Parallel accumulators
- f. Loop unrolling ✔ Correct!
- g. Strength reduction

Your answer is correct.

This is an example of loop unrolling.

The correct answer is: Loop unrolling