Midterm Exam   CS 600

Name: Lan Chang

Question 1

Proof:

The definition of Ω is: $f(n)$ is $\Omega(g(n))$ if there is a constant $c>0$ and an integer constant $n_0\geq1$ such that $f(n)\geq c\times g(n)$ for $n\geq n_0$.

In this question, $f(n)=n$ and $g(n)=n^2$.

For $c>0$ and $n_0\geq1$, $c\times n^2$ is always larger than n at last, and we cannot find the c and n_0 that $n\geq c\times n^2$.

As a result, n is not $\Omega(n^2)$.

Question 2

Proof:

The definition of o is: $f(n)$ is $o(g(n))$ if, for any constant $c>0$, there is an integer constant $n_0\geq0$ such that $f(n)\leq c\times g(n)$ for $n\geq n_0$.

In this question, $f(n)=n$ and $g(n)=n^2$.

We can choose $c=1$ and $n_0=1$, and if $n\geq1$, $n\leq n^2$.

As a result, n is $o(n^2)$.

Question 3

Pseudo code:

```
Algorithm factorial(n)
  if n=0 then
    return 1
  if n=1 then
    return 1
  else
    return n*factorial(n-1)
```

Question 4

Answer:

In this algorithm, we should apply stable bucket sort three times.

First, we should sort these triplets using the third component m.

Next, we should sort these triplets using the second component l.

Then, we should sort these triplets using the first component k.

Finally, we get these triplets ordered.

There are 3 numbers in each triplet, and we assume there are n numbers of triplet, as a result, the running time is $O(3\times(n+N))=O(n+N)$.

Question 5

Answer:

In this algorithm, we should traversal this matrix like the stairs.

Step 1, we choose the position (1, 1) as the beginning.

Step 2, we should go right at this row until we find the element is not 1, and we should stop and record this row number, then we should go down at this column until we find the element is 1, and we should stop and choose this element as the beginning.

Step 3, we should repeat the step 2 until we cannot continue to go right and go down (if we are at the last row and we still can go right to find more 1 at the last row, we will continue). And the final recorded row number is the answer.

In this method, we traversal this matrix without any going left or going up, so the longest path distance is 2n, which is from (1, 1) to (n, n). As a result, the running time for this algorithm is $O(2n)=O(n)$.

Question 6

Answer:

In this algorithm, we should use radix sort.

Generally, the base of integers is 10. However, in this question, we can change this base into n. We can transform each integer into the number in base n, and change their form into (a, b). We can divide these integers by the base n, and a is the quotient the and b is the remainder. Because these integers are both smaller than n^2-1 and $(n^2-1)/n<n$, these

two numbers are both smaller than n. For example, 15 in base 10 is (1, 5) or 15 in base 12 is (1, 3) or 15 in base 18 is (0, 15).

Then, we can use radix sort to sort these tuples. We sort these tuples using the second component b and the first component a successively.

Finally, we get the final result ordered.

In the transforming step, we just divide n numbers by the base, the running time is O(n). In the sort step, due to the radix sort theorem, a and b are both in the range [0, n-1], so the running time is O(2×(n+n))=O(n). As a result, the total running time is O(n).

Question 7

Answer:

First, we should sort these tasks by the finish time and schedule task 1 on machine.

Second, we should consider these tasks successively. For each task, if its start time is later than those tasks' finish time we have scheduled before on machine, we will schedule this task on machine. Otherwise, if this task doesn't fit this condition, we will consider the next task. Repeat this process until we have considered all the tasks in T.

Finally, we will get the schedule.

Proof:

If we want to prove that this algorithm is the best optimal greedy solution, we must prove the following two propositions.

Proposition 1: for task t(i), the finish time given by this algorithm f(i) ≤ the finish time given by some other optimal solutions g(i).

We can prove this statement by induction method.

If n=1, it is obviously true because there is only one task.

We assume that if n=i-1, f(i-1)≤g(i-1) is true.

If n=i, due to this algorithm, we will choose the task which has the minimum finish time, so f(i)≤g(i).

As a result, this proposition has been proved.

Proposition 2: the schedule given by this algorithm n = the schedule given by the best optimal solution k.

We can prove this statement by simple contradiction argument.

We can assume that n<k. Due to proposition 1, f(n)<g(n). If n<k, there must be some tasks which start after f(n) and g(n). However, due to this algorithm, we will not stop after select the task n, and we will continue to select more tasks on machine. As a result, there is a contradiction and n≥k. And k is the best solution, so n=k.

As a result, this proposition has been proved.

In conclusion, this algorithm is the best optimal solution.

Running time:

In the sort process, we can use the merge sort, and the running time is O(nlogn).

In the schedule process, we will consider each task, and the running time is O(n).

As a result, the total running time is O(nlogn).

Question 8

Answer:

In this algorithm, we should use divide and conquer algorithm. The main idea is that we divide these teams into 2 parts. We assume that they have played with each teams in their own part and we need to let them play with each teams in the other part. And finally, all the team have played with each other. We can continue divide each part into 2 parts and repeat the above process, until there are only two teams, and we can let them play once.

Step 0: we need to construct an n×n matrix, and we fill the first column with 1, 2, 3, 4…, n-3, n-2, n-1, n.

Step 1: we divide the n teams into n/2 parts and each part has 1×2 elements. In each part, we swap these two elements and fill the second column with 2, 1, 4, 3…, n-2, n-3, n, n-1.

Step 2, we divide the first two columns into n/4 parts and each part has 2×4 elements. In each part, we swap the first 2×2 matrix with the second 2×2 matrix and fill the third and fourth columns with these elements like 3, 4, 1, 2…, n-1, n, n-3, n-2 in the third column and 4, 3, 2, 1…, n, n-1, n-2, n-3 in the fourth column.

Step i, we divide the first 2^{i-1} columns into $n/(2^i)$ parts and each part has $2^{i-1}\times 2^i$ elements. In each part, we swap the first $2^{i-1}\times 2^{i-1}$ matrix with the second $2^{i-1}\times 2^{i-1}$ matrix and fill the $2^{i-1}+1$ to 2^i columns with these elements.

Finally, if i=log n, we finish and the final matrix is the schedule. The column 1 is the team and other columns are their opponents in round i (i=column number-1).

We must consider each element in this matrix. As a result, the total running time is $O(n^2)$.