Review question

(1) A good program is defined as a program that runs correctly and it is easy to read and understand, debug and modify.

(2) 'Data' means a value or set of values. A 'record' is a collection of data items. And 'file' is a collection of related records. 'Primary key' is a unique data item that uniquely identifies the record in the file.

(3) As people organize their files, there are some data structures that organizes and saves file in programming. These are building blocks of a program. so programmer must choice most appropriate data structures for a program. For example stack is similar to stacking the bowls and keep them.

(4) There are two ways to categorize data structure. First is primitive and second is non-primitive. primitive data structures are the fundamental data types that are supported by a programming language. Non-primitive data structures are created using primitive structures. And non-primitive structures can be classified into linear and non-linear data structures.

(5) The application of an appropriate data structure provides the most efficient solutions. Appropriate data structure is more useful to solve the problem within the required constraint(time and space).

(6) 'Traversing' is to access each data item exactly once so that it can be processed. 'Searching' is to find the location of data item that satisfy certain condition. 'Inserting' is add new data items to given list of data items. 'Deleting' is to remove a particular data item to given collection of data item. 'Sorting' is a process of arranging all data items in a particular order(ascending or descending). 'Merging' is a process to combine two sorted data items into a list of sorted data items.

(7) An array and a linked-list are linear data structures. Compared with linked-list, array has fixed size and insertion and deletion of elements can be problematic because of shifting of elements. But linked list may need more memory space than array.

(8) Abstract data type is the structure that focuses on what it does and ignores how it does its job. It means the structure do not care about implementation o f data.

(9) There are 6types of Data structures: Arrays, Linked lists, Stacks, Queues, Trees, Graphs. **Array** is a collection of similar data elements. so data elements of arrays have same data type. Array is useful to store large amount of similar type of data but it is of fixed size. And it can have problem because of shifting of elements from their positions. **Linked-list** is a flexible sequential list that uses nodes. Every nodes in the list points to the next node. Node is consisted of the value of the data and a pointer to the next node in the list. If you want to add a new node, you can assign to the list, so it is easy to insert or delete data elements. **Stack** is a linear data structure where insertion and

deletion of elements are done at only one end(top). Because stack supports three basic operations, we can easily add or remove elements of stacks. But stack isn't flexible and it lacks scalability. Queue is linear data structure. With the queue, the new pieces of data are placed at the rear of the data structure, and the deletions are placed at the front. Queue have advantage of being able to handle the data flexible and fast. On the other hand, when we use the queue as a data structure, there may be such a situations that the queue has remain but we can't insert a new element. **Tree** is a non-linear data structure that consists of a collection of nodes arranged in a hierarchical order. Advantage of using tree as data structure is providing quick search, insert, and delete operations. But it has complicated deletion algorithm. **Graph** is a non-linear data structure which is a collection of vertices. A graph is consisted of nodes and edges. Graph is best models that reflects real-world situations but some algorithms are slow and very complex.

(10) The definition of algorithm is a procedure for performing some calculation. This means an algorithm provides a blueprint to write a program to solve a particular problem. For example it is an algorithm that determines if it is the same as my ID.

ex) step 1: input first ID as A
step 2: IF A==sam
print "It's correct"
else
print "It isn't correct"
[END OF IF]
step 3: END

(11) There are two ways to approaches to designing an algorithm: Top-down approaches and Bottom-up approaches. Top-down approach is dividing the complex algorithm into one or more modules. On the other hand, Bottom-up approach is starting from most basic or concrete modules and then towards higher level modules.

(12) Modularization is a function of top-down approach. It means through decomposing the algorithm into manageable modules, it makes stepwise refinement possible. Modularization is useful for generation of test cases, implementation of code, and debugging.

(13) A tree is a non-linear data structure which consists of a collection of nodes arranged in a hierarchical order. One of the nodes is designated as the root node, and the remaining nodes can be partitioned into disjoint sets such that each set is a sub-tree of the root.

(14) we can understand connection between workstation and network by using graph. And also These have many applications in computer science and mathematics such as we can find the shortest path between the nodes of a

graph.

(15) I think two important things in Choosing an algorithm to solve a particular problem are space and time. The time is basically the running time of a program and the space is the amount of computer memory that is required during the program execution.

(16) It is difficult to create an ideal algorithm that kept the balance. When you choose a program, first you have to find what is the major constraint and you have to sacrifice one of them after deciding which one to focus on, time or space.

(17) The efficiency of an algorithm is expressed in terms of the number of elements that has to be run on program and kinds of the loop that is being used. Depending On loops or recursion, the efficiency of algorithm will be changed.

(18) I will use the big O notation. It is concerned with what happens for very large value of n. For example, if algorithm performs n operations to sort n elements, then that algorithms would be described as an O(n) algorithms.

(19) The significance of big O notation is Big O notation allows programmer to analyze algorithms in terms of overall efficiency and scalability. The limitation of big O notation is the big O analysis only tell us how the algorithms grows with the size of the problem, not how efficient it is, as it does not consider the programming effort.

(20)**Worst-case running time** gives us an upper bound that the algorithm will never go beyond this time limit. **Average-case running time** gives us the expected behaviour of the algorithm when the imput is randomly drawn from a given distribution. **Best-case running time** is used to analyse an algorithm under optimal conditions. **Amortized running time** gives us average performance of each operation in the worst case.

(21) **Constant time algorithm** : running time complexity given as O(1), **linear time algorithm** : running time complexity given as O(n), **logarithmic time algorithm:** running time complexity given as O(log n), **polynomial time algorithm** : running time complexity given as O(n^k) where k>1, **exponential time algorithm** : running time complexity given as O(2^n).

(22) Examples of functions in O(n^2) include : $100n^2$+30
Examples of functions not in O(n^2) include : n^3

(23) The little o notation provides a non-asymptotically tight upper bound for f(n).

(24) Examples of functions in o(n^2) include : $n^{1.8}$, n
Examples of functions not in o(n^2) include : $100n^2$, $10n^2$+13

(25) While big O notation is asymptotically tight upper bound for f(n), little o notation is non-asymptotically tight upper bound for f(n). And While big O notation can have some c>0, little o notation should be any c>0.

(26) The Omega notation provides a tight lower bound for f(n). This means that the function can never do better than the specified value but it may do worse.

(27) Examples of functions in $\Omega(n^2)$: $5n^2$+10, n^3

Examples of functions not in $\Omega(n^2)$ include : 10n, 3n+5

(28) Theta notation provides an asymptotically tight bound for(n). The best case in Θ notation is not used so when we simply write Θ it means same as worst case Θ.

(29) Examples of functions in $\Theta(n^2)$: n^2+2n+1

Examples of functions not in $\Theta(n^2)$ include : n^3

(30) Little Omega notation provides a non-asymptotically tight lower bound for f(n). This is unlike the Ω notation where we say for some c>0.

(31) Examples of functions in $\omega(n^2)$: n^3

Examples of functions not in $\omega(n^2)$ include : n^2, n

(32) Little Omega notation is unlike the Ω notation where we say for some c>0(not any). c is positive constants that coefficient of g(n).

(33)

$0 \leq h(n) \leq cg(n)$

substituting n^2+50n as h(n) and n^2 as g(n)

$0 \leq n^2+50n \leq n^2 \bullet c$

dividing by n^2

$$0 \leq 1+\frac{50}{n} \leq c$$

Now to determine the value of c, we see that 50/n is maximum when n=1 therefore c=51.

To determine the value of n_0

$$0 \leq 1+\frac{50}{n_0} \leq 51$$

$$-1 \leq \frac{50}{n_0} \leq 50$$

$-n_0 \leq 50 \leq 50 \bullet n_0$

so n_0=1

hence $0 \leq n^2+50n \leq 51n^2 \quad \forall n \geq n_0 = 1$

(34)

$0 \leq h(n) \leq cg(n)$

substituting $3n^2$ as h(n) and n^3 as g(n)

$0 \leq 3n^2 \leq n^3 \bullet c$

dividing by n^3

$$0 \leq \frac{3}{n} \leq c$$

Now to determine the value of c, we see that 3/n is maximum when n=1 therefore c=3.

To determine the value of n_0

$0 \leq \frac{3}{n_0} \leq 3$

$0 \leq 3 \leq 3n_0$

so n_0=1

hence $0 \leq 3n^2 \leq 3n^3 \quad \forall\, n \geq n_0 = 1$

(35)

$0 \leq h(n) \leq cg(n)$

substituting n^3 as h(n) and n^2 as g(n)

$0 \leq 3n^3 \leq n^2 \bullet c$

$0 \leq n \leq c$

hence $n^3 \neq O(n^2)$

(36)

$0 \leq cg(n) \leq h(n)$

substituting $\sqrt{n}$ as h(n) and $\log n$ as g(n)

$0 \leq \log n \bullet c \leq \sqrt{n}$

dividing by $\log n$

$0 \leq c \leq \frac{\sqrt{n}}{\log n}$

$\lim_{n\to\infty} \frac{\sqrt{n}}{\log n} = \infty$

$0 \leq c \leq \infty$

hence $\sqrt{n} = \Omega(\log n)$

(37)

$0 \leq cg(n) \leq h(n)$

substituting n^2 as h(n) and 3n+5 as g(n)

$0 \leq n^2 \bullet c \leq 3n + 5$

dividing by n^2

$0 \leq c \leq \frac{3n+5}{n^2}$

$\lim_{n\to\infty} \frac{3n+5}{n^2} = 0$

but c>0

hence, $3n + 5 \neq \Omega(n^2)$

(38)

$0 \leq c_1 \bullet g(n) \leq h(n) \leq c_2 \bullet g(n)$

substituting $\frac{1}{2}n^2 - 3n$ as h(n) and n^2as g(n)

$0 \leq c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2$

dividing by n^2

$0 \leq c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$

Now to determine the value of c_2, we see that 3/n is maximum when n=1 therefore $c_2 = \frac{1}{2}$.

To determine c_1 using Ω notation, we can write

$0 < c_1 \leq \frac{1}{2} - \frac{3}{n}$

we see that $0 < c_1$ is minimum when n=7 therefore, hence $c_1 = \frac{1}{14}$

Thus, in general, we can write, $\frac{1}{14}n^2 \leq \frac{1}{2}n^2 - 3n \leq \frac{1}{2}n^2$

Multiple-choice Questions

1.(a)
2.(c)
3.(a)
4.(b)
5.(a)
6.(b)
7.(a)
8.(d)
9.(d)
10.(b)
11.(b)
12.(a)
13.(b)

True or False

1.F
2.T
3.F
4.F
5.F
6.T
7.F
8.F
9.F

10.F
11.T
12.T
13.F
14.F
15.F

Fill in the Blanks
1. Data structure
2. Algorithms
3. linear data structure
4. Data type
5. root=NULL
6. considered apart from the detailed specifications or implementation
7. input size
8. Amortized
9. index
10. TOP
11. peep
12. we try to insert an element into stack or queue that is already full
13. Queue
14. rear, front
15. linear data structure
16. algorithm
17. time complexity
18. average-case running time
19. O(1)
20. module
21. A top-down
22. Worst
23. omega
24. non-asymptotically
25. =