UNIVERSITY OF BUEA
FACULTY OF ENGINEERING
END-OF-SEMESTER EXAMINATIONS

TTN

DEPARTMENT: Computer Engineering
MONTH: March
YEAR: 2014
DATE: 04/03/2014 TIME: 15:00-18:00
TIME ALLOWED: 3 hours

COURSE INSTRUCTOR: Mr. Nyanga B.
COURSE CODE & NUMBER: CEF405
COURSE TITLE: Analysis and Design of Algorithm.
CREDIT VALUE: 4

INSTRUCTIONS: **Read through EACH question before you answer it. Follow instructions for EACH Section. Time is allocated for a MAXIMUM POSSIBLE MARK of 70. Programs assumed to be in Standard Prolog. State any assumptions made. Penalty for poor English or poor presentation of work.**

Section I. write True/false:

1. $O(n^2) = O(2n^2)$
2. $O(n^2) = O(100n)$
3. $O(50n) = O(n/2)$
4. $O(n) = O(\lg n)$
5. $O(\lg n) = O(\lg n + 1000)$
6. Algorithm analysis provides us a single formula by which to determine the best data structure to use
7. $O(\log n)$ problems are considered intractable.
8. $O(2n)$ problems are intractable.
9. An algorithm with an $O(\log n)$ running time is considered fast.
10. A binary search should be faster than a linear one.
11. The Bubble sort algorithm makes on the order of n passes through an n-element array.
12. The binary search is an efficient way to search any large array
13. A linear search must inspect, on average, about half the elements in an array.
14. To sort an array normally requires swapping data items that are in order.
15. It is easy to prove correctness of a program that works.

Section II: circle the letter with the correct answer.

1. Where a problem can be broken down into two problems, one of which is simple to solve and the other is a smaller instance of the original problem, one way to solve the problem is
(a) greedy; (b) object-oriented; (c) structured; (d) recursive
2. Divide-and-conquer refers to the use of
(a) intractability; (b) recursion; (c) dominant expressions; (d) branching; (e) arithmetic operators
3. A recurrence defines a function
(a) selectively; (b) iteratively; (c) exponentially; (d) algorithmically; (e) redundantly
4. A program's correctness (a) may be proven through testing;
(b) may be proven mathematically; (c) may be proven through a combination of testing and persuasive language;
(d) can never be proven; (e) none of the above
5. An *assert* statement is used to
(a) perform a search; (b) help detect syntax errors; (c) help detect logic errors; (d) help detect user-input errors;
(e) document program code
6. In a correct algorithm, a loop invariant is true
(a) always; (b) never; (c) at the start of a loop body; (d) throughout execution of loop body; (e) sometimes
7. In an inductive proof, showing that $P(0)$ is true is
(a) the base step; (b) the inductive step; (c) unnecessary; (d) sufficient to prove $P(x)$ implies $P(x+1)$; (e) sufficient to prove $P(x)$ for all x
8. To prove that an algorithm terminates, we usually rely on
(a) recursion; (b) *break* statements; (c) convergence; (d) total correctness; (e) partial correctness
9. Total correctness is proven by showing
(a) partial correctness; (b) termination; (c) good test results; (d) partial correctness and termination; (e) termination and good test results
10. An inductive proof argues in part that
(a) if a certain assertion is true for n then it is also true for $(n+1)$; (b) a certain assertion is not true for any value n; (c) a certain assertion leads to a contradiction; (d) if one assertion is not true, then a second assertion must be true
11. In Quicksort, the pivot is
(a) a subscript; (b) an element value; (c) a turning point in the execution of the algorithm; (d) used to leverage performance; (e) none of these

12. What is the fastest sure way to search for a value in an unsorted array of numbers?
(a) calculate hash value;
(b) scan from beginning to end until found; (c) sort and perform binary search; (d) choose random elements until the number is found; (e) no fastest way exists.
13. To efficiently locate the number 9 in an array of random integers, we would use a
(a) linear search; (b) binary search; (c) Bubble sort; (d) Quick sort; (e) none of these
14. The analysis of algorithms is most concerned with
(a) testing solutions; (b) the documentation of programs; (c) object orientation; (d) the time complexity of programs; (e) all of these
15. For a problem of size n, a solution of the *worst* time complexity would be
(a) 1 (constant time); (b) n (linear); (c) n squared (quadratic); (d) 2 to the n power (exponential); (e) none is worst
16. Each step of the binary-search algorithm
(a) reduces the size of the sub-array to be searched by about half; (b) finds the search key; (c) reports failure; (d) moves one array element; (e) compares two array elements
17. The time complexity of the binary search is on the order of
(a) 1; (b) n; (c) log2(n); (d) n squared; (e) 2 to the nth power

$2n+1 = c\,g(n)$
$\alpha\, g(n)$

18. Which sort has an execution time proportional to $n \times \log 2(n)$?
(a) insertion sort; (b) bubble sort; (c) selection sort; (d) insertion sort; (e) Quicksort

$2n+1 \quad O\, g(n)$
$3n$

19. Where f is a function, O($f(n)$) means _______ $f(n)$.
(a) exactly; (b) at most; (c) roughly proportional to; (d) at least; (e) following a specification

$T(n)$

20. A merge algorithm
(a) takes longer than Quicksort; (b) performs a search; (c) requires two or more sorted arrays; (d) all of these; (e) none of these
21. Big-O notation sets
(a) a precise time complexity; (b) a lower bound; (c) an upper bound; (d) an upper and lower bound; (e) an indefinite estimate
22. Theta notation sets
(a) a precise time complexity; (b) a lower bound; (c) an upper bound; (d) an upper and lower bound; (e) an indefinite estimate
23. Big-Omega notation sets
(a) a precise time complexity; (b) a lower bound; (c) an upper bound; (d) an upper and lower bound; (e) an indefinite estimate
24. Asymptotic analysis relates most closely to the
(a) growth of processor speeds; (b) decline in unprocessed data; (c) rate of growth of functions; (d) complexity of source files; (e) constant factors that affect speed

Section III: answer all questions for a maximum of 31 marks

1. Find the solution, in big-O notation, for $T(n)$ =
(a) {1 if $n = 0$; $2 + T(n-1)$ otherwise}
(b) {1 if $n \leq 2$; $2 + T((n-1)/2)$ otherwise} (6 + 6=12marks)
2. The recursive definition of Fibonacci numbers gives us a recursive algorithm for computing them:

```
FIB(n)
 if (n < 2)
 then return n
 else return FIB(n − 1) + FIB(n − 2)
```

(i) Draw the recursive tree for F(8). What is the problem with this approach? List the number of times F(i) for 1< i < 7 is computed.
(ii) What do you understand by *memorization?*
(iii) What are the basic properties used in dynamic programming? With these properties; design a more algorithm for the Fibonacci numbers. ((5+3+5)+2+(2+3) = 20mrks)

fib(n) = memo fib(1) + memo fib(2)

fib(n) = memo fib - fib()