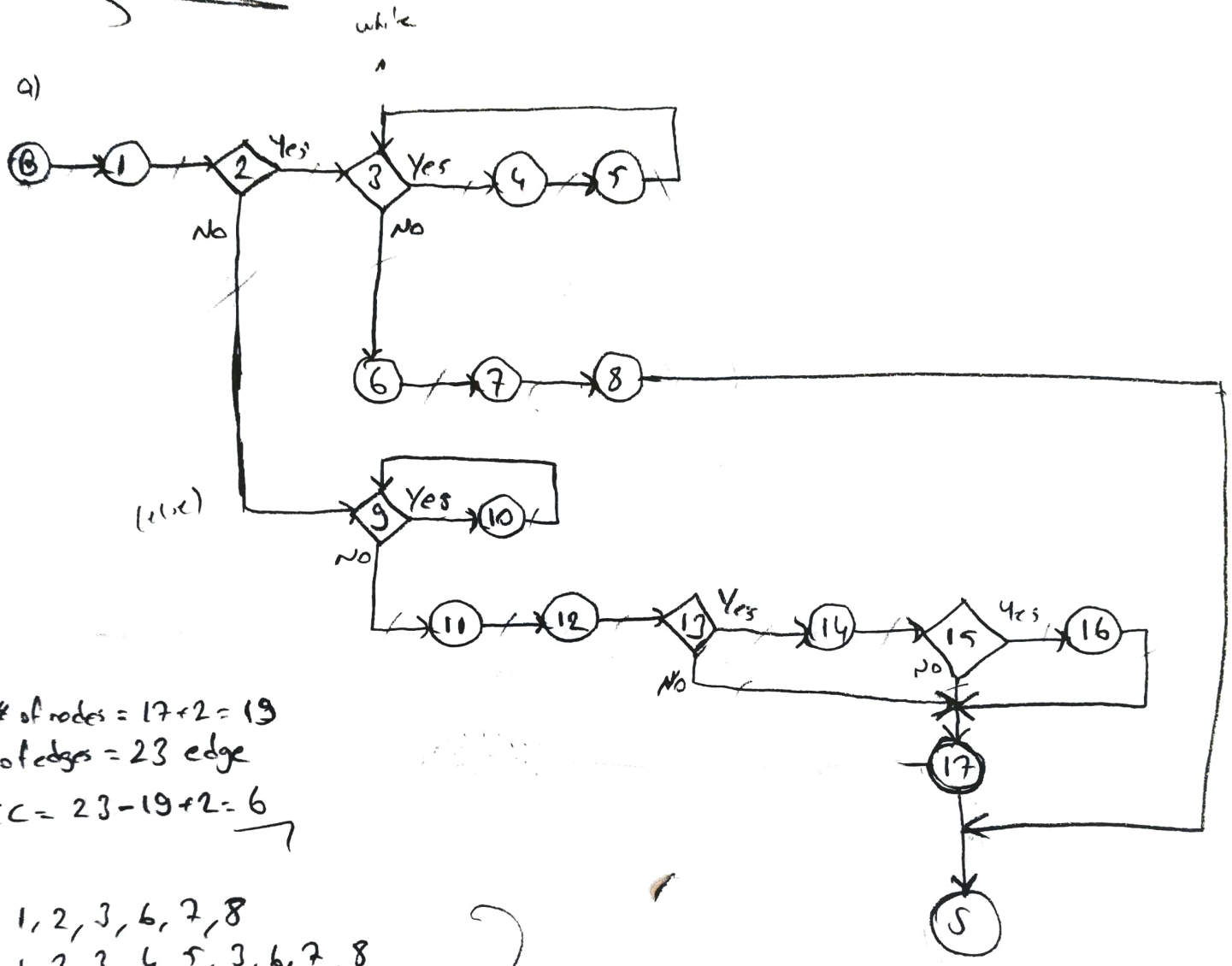
On my honor, I have neither given nor received any unauthorized and/or inappropriate assistance for all sessions of this exam. The work done on this exam is totally my own. I understand that by the school code, violation of these principles will lead to an F grade and is subject to harsh discipline issues.

Yunus Emre Erturg
150117064

Q1) a)



b)

# of nodes = 17+2 = 19
# of edges = 23 edge
CC = 23-19+2 = 6

c) 1, 2, 3, 6, 7, 8
1, 2, 3, 4, 5, 3, 6, 7, 8
1, 2, 9, 11, 12, 13, 17
1, 2, 9, 10, 9, 11, 12, 13, 17
1, 2, 9, 11, 12, 13, 14, 15, 17
1, 2, 9, 11, 12, 13, 14, 15, 16, 17

} 6 independent paths

d) i) statement coverage = 8/17
ii) branch coverage = 10/23

e) i) 1) s.c = 9/17
2) b.c = 10/23
ii) 1) s.c = 7/17
2) b.c = 8/23

Function Points:

→ The function point count is modified by complexity of the project.
→ FP's can be used to estimate LOC depending on the average number of LOC per FP for a given language.
→ FP's are very subjective.

- Object Points (Application Points):

→ Are an alternative function-related measure to func. points when 4GL's or similar lang are used.
→ Object points are not same as object classes.
→ Object points are easier to estimate.
→ They can be estimated at a fairly early point in the development process.

- Factors Affecting Productivity:

Application domain experience, process quality, project size, Technology support, Working environment.

- Estimation Techniques:

Algorithmic cost modelling, expert judgement, estimation by analogy, Parkinson's Law, Pricing to win.

Pricing To Win:

→ The project costs whatever the customer has to spend on it.
→ Advantage: You get the contract.
→ Disadvantage: The probability that the customer gets the system he or she wants is small. Costs do not accurately reflect the work required.

Top down and Bottom up Estimation:

→ Top down: start at the system level and assess the overall system functionality and how this is delivered through sub systems.
→ Bottom up: start at the component level and estimate the effort required for each component. Add these efforts to reach a final estimate.
Top down: usable without knowledge of the system architecture.
Bottom up: usable when the architecture is known.

Pricing To Win:

→ This approach may seem unethical and unbusinesslike however, when detailed information is lacking it may be the only appropriate strategy.

Algorithmic Cost Modeling:

→ Cost is estimated as a mathematical function of product, project and process attributes whose values are estimated by project managers.
→ The most commonly used product attribute for cost estimation is code size.

Estimation Accuracy:

→ The size of a software system can only be known accurately when it is finished.

$Effort = AVC * \text{number of function points}$

$Effort = A \times Size^{B} \times M \quad A = 2.94$

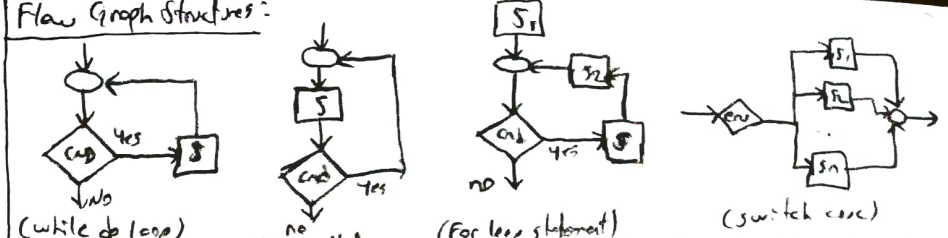
$Dev = 3 \times PM^{0.33+0.2*(B-1.01)}$

$B = \left(\frac{\text{Scale factors}}{100}\right) + 1.01$

M = multiplier say $1 \times \ldots$

$\frac{b-k}{k} \times 100$

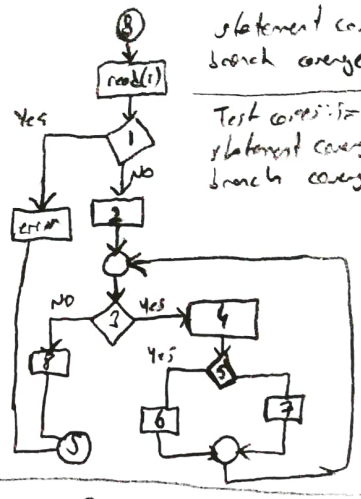
Flow Graph Structures:



(while do loop) (do while) (For loop statement) (switch case)

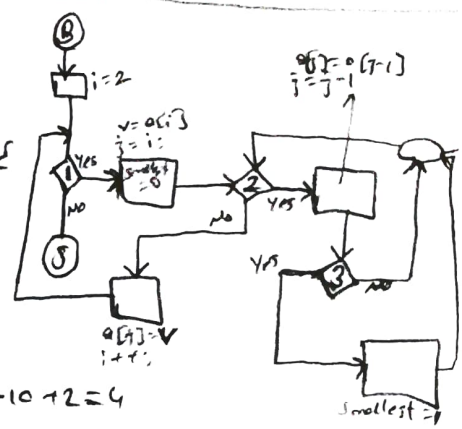
```
• read(i);
• if((i<0)||(i>100)) ①
  - error();
• else
  - {sum=0; x=0; ②
  - while(x<i) ③
    • {x=x+1; ④ ⑥ ⑤
    - if(i==10)sum=1;
      else sum = sum+x; }④
  - print(sum) }
```

Test cases: i = -1, 0, 1, 50, 99, 100, 101
statement coverge = 9/10 = %90
branch coverge = 12/14 = %86



Test cases: i = 1, 1, 10
statement coverge = 10/10 = %100
branch coverge = 14/14 = %100

```
void insertion sort(){
  int i,j,v;
  int smallest;
① for(i=2;i<=n;i++){
    v=a[i]; j=i; smallest=0;
② while(a[j-1]>v && !smallest){
③     a[j]=a[j-1]; j=j-1;
      if(j<=1)
        smallest=1;
    }
    a[j]=v
  }
}
```



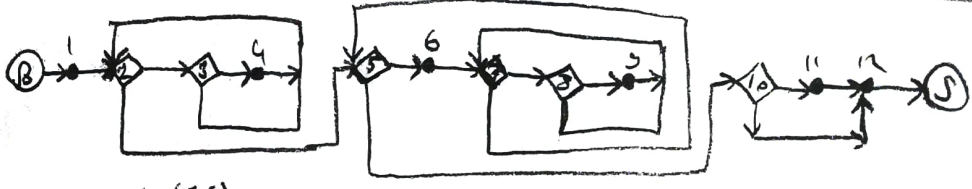
CC = 12 - 10 + 2 = 4

```
void Topo Sort(){
  Queue Q; int ctr=0; Vertex v,w;
  Q=create Queue(NumVertex); (1)
  for each vertex v (2)
    if(indegree[v]==0) enqueue(v,Q); (3,4)
  while(!isempty(Q)){ (5)
    v=dequeue(Q); topnum[v]=++ctr; (6)
    for each w adjacent to v (7)
      if(--indegree[w]==0)enqueue(w,Q); (8,9)
  }
  if(ctr != NumVertex) reportError('graph cyclic') (10,11)
  free Queue (12)
```

$10 * 10^4 + 5 = 2005$ cases

Black Box

1. Specify input categories:
   A) Number of nodes in the graph
   B) Number of edges in the graph
   C) Types of connectedness
   D) Type of graph
2. Divide categories into ECs
   A) 0, 1, 2, 3, 4, 5, 20, 50, 100, 1000, 10000 (10 case)
   B) 0, 1, 3, n, $n^2/4$, $n^2/2$, $3n^2/4$, n^2-1, n^2 (10 case)
   C) Not connected, sparsely connected, densely connected, fully connected (4 case)
   D) bus, ring, star, mesh and hybrid (5 case)



N = 12 + 2 = 14 (BS)
E = 19
CC = 19 - 14 + 2 = 7

independent paths:
1) 1 2 5 10 12
2) 1 2 5 10 11 12
3) 1 2 5 6 7 8 7 5 10 12
4) 1 2 5 6 7 5 10 12
5) 1 2 5 6 7 8 9 7 5 10 12
6) 1 2 3 2 5 10 12
7) 1 2 3 4 2 5 10 12

dependent paths
1) 1 2 5 6 7 5 10 11 12 (2,3)
2) 1 2 5 6 7 8 7 5 10 11 12 (2,4)
3) 1 2 5 6 7 8 9 7 5 10 11 12 (2,5)
4) 1 2 3 4 2 5 6 7 8 9 7 5 10 11 12 (2,5,7)

Testing:

- Dimensions of sw quality: Adaptability, ability to change and Operational characteristics
- System Properties: Portability, Reusability, Interoperability, Maintainability, Flexibility, Testability, Correctness, reliability, efficiency, Integrity, Usability
- Validation: building right product.
- Verification: building the product right.

Principles of sw testing:

1- Testing is a process of executing a program with the intent of finding a defect
2- A good test case is one that has a high probability of finding defect.
3- A successfull test is one that uncovers an as yet discovered defect.
4- Testing cannot show absence of defect.!
5- Successfull testing shall be followed by a separate debugging phase.

Jargon of Testing:

- Error: A mistake (human action) made by developer.
- Defect: A difference between the incorrect program and its correct version.
- failure: incorrect result of computation.

Facts of Testing: V+V engineer define correct incorrect

- Full automation of testing is impossible (4 reasons)
1- the total behaviour of a program is undecidable.
2- Exhaustive testing is intractable
3- tracking of failures to errors is impossible.
4- We can never be sure testing tools work perfectly.

Module (unit) Testing:

1- Each independent unit tested separately
2- level: source code
3- need for simulated execution environment.

Integration Testing:

- Modules grouped into subsystems.
"big bang" all the modules tested as whole.

System testing:

- the whole system tested.
- target: system properties (performance, capacity...)

Volume Testing: a non functional test. Refers to testing software application with a certain amount of data.

Security testing: Determines that a sw product protects data and maintains functionality as intended.
→ Confidentially, Integrity, Authentication, Authorization, availability, Non repudiation

Performance Testing: Determines how a system performs in terms of responsiveness and stability under a particular workload

Black Box (Functional) Testing: internal details of system are hidden and cannot be studied from outside. based on input classification

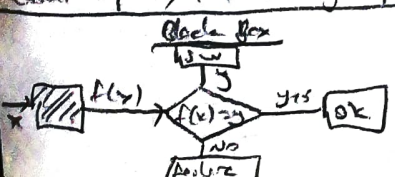
White Box (structural) Testing: structure of software is examined in detail at the level of program code.

Management of Testing:

→ Plan, execute, evaluate, document, report.

Standard for Soft. Test Documentation:

- Test Plan, Test Design Specification, Test case specification, Test procedure specification, Test item transmittal report, test log, test incident report, test summary report.



Black Box Testing:

- Principles:
→ Based on specifications and documents
→ Code not necessarily needed
→ Applies especially to integration testing, system testing, acceptance testing.

- Domain Partitioning: Equivalence Classes
→ system domain: set of all input values
→ Equivalence class: certain set of input values

- Equivalence Classes:
→ Each EC represents a central property of the system
→ Each value in EC makes system behave in the same manner
→ Based on system's specification and experience/intuition of tester.
→ for testing purposes, one input from each EC is enough.
- system shall be tested with several input values from each EC (in practice)

- Boundary Conditions:
→ open boundaries (<, >)
→ closed boundaries (=, ≤, ≥)
→ on point
→ off point

- The Category-partition method:
1→ Specification of input categories
2→ Division of categories into choices = EC
3→ Test specification
4→ Generation of test cases for the test frames into executable form
5→ Storing the testware into a test database
6→ Testing of the unit by the test cases

- System Testing / GUI Testing:
→ Target: operations available at the UI.
→ Parameters of operations divided into EC.
→ Testing by all different combinations of EC
→ testing of operation sequences (not independent)

- How Many Tests?:
→ # of (independent) combinations = Total number of tests.
→ $E_1 * E_2 * \ldots E_k$ E_i = # EC for parameter i.

White Box Testing:

- Principles:
→ Details of source code analyzed.
→ Design of test cases on the basis of code structure.
→ Execution Path: a certain sequence of program statements.
→ Control flow testing: based on the execution order of the statements.
→ Data flow testing: based on the processing of the data during execution.
→ control (flow) graph: abstraction of program's control flow, in graphical form
→ data flow graph: abstraction of program's data flow, usually extension of control graph.
→ coverage: the relative amount of statements executed during testing.

- Cyclomatic Complexity (CC) of a piece of code is the number of linearly independent paths through the code
→ with no decision structure: CC = 1 single path
$CC = E - N + 2$, E = edge, N = node

Case Studies for BB and WB Testing:

- First case: Insertion Sort:

→ Steps of Category-Partition Method:
→ Specify input categories
→ Divide categories into EC
→ Determine test cases
→ Generate test cases for the test frames into executable form
→ Store the testware into a test database
→ Apply test.

1. Specify Input Categories:

A) Size of array D) min element value
B) Types of elements E) Position of max element
C) Max element value F) Position of min element

2. Divide Categories into EC's:

A) Size of array ⇒ negative, 0, 1, 3, 101, 100000 (6 cases)
B) Types of elements ⇒ Integer, non integer (2 cases)
C) Max and min ⇒ large neg., -1, typical, 1, large pos (5 cases each)
D) Position of max (min) ⇒ start, middle, end, out of bound (4 cases each)

* Important Questions:
- Number of exhaustive combinatory test cases?
$6 * 4 * 4 * 5 * 5 * 4 * 4 = 9600$ cases (too many)
- what you should do in case the number of exhaustive combinatory test cases is infeasible
→ use optimizing (cover each NV class by a test case, and each other EC by at least one test case) and extending (cover each parameter in a test case, in the set of pairs, triplets) principles.

- Two test case examples:
→ (101, integer, -5020, 7101, 2, 47)
→ (3, float, 0, 100, 2, 3)

→ Test By WB Testing:

1. Draw control flow graph of the unit
2. Compute cyclomatic complexity (CC)
3. Determine independent paths (IPs)
4. Decide on the coverage (statement, branch, condition, multi condition, path)
5. Prepare test cases regarding coverage

Software Cost Estimation

- Software cost components:
→ Hardware and software costs
→ Travel and training costs
→ Effort costs (dominant factor)

- Software Pricing Factors
→ market opportunity, cost estimate uncertainty, contractual terms, Requirements volatility, financial health

Productivity Measures:
→ size related measures based on some output from the software process. This may be lines of delivered source code.
→ Function related measures based on an estimate of the functionality of the delivered software. Function points are the best known of this type

Measurement Problems:
→ Estimating the size of the measure.
→ Estimating the total number of programmer
→ Estimating contractor productivity.

Lines of code:
→ This model assumes that there is a linear relationship between system size and volume of documentation.

Productivity Comparisons:
→ The lower level the language, the more productive the programmer
→ The more verbose the programmer, the higher the productivity.