

Big-O: denote the **worst-case** runtime of an algorithm.

* O(1) — Constant runtime — amazing
* O(log n) — Logarithmic runtime — good
* O(n) — Linear runtime — alright / good
* O(n log n) — Linearithmic runtime — bad
* O(n^2) — Quadratic runtime — TERRIBLE
* O(2^n) — Exponential runtime — TERRIBLE
* O(n!) — Factorial runtime — TERRIBLE

Sorts – Bubble / insertion / quick / merge

Fisher yates vs naïve

Recursion (know an example)

Name, image or description:

* Perlin noise
* Travelling salesman problem
* Gale/Shapley (Stable Marriage)
* HCF (Euclid’s Algorithm)

BREADTH FIRST VS DEPTH FIRST SEARCH –

[https://stackoverflow.com/questions/687731/breadth-first-vs-depth-first](https://stackoverflow.com/questions/3332947/when-is-it-practical-to-use-depth-first-search-dfs-vs-breadth-first-search-bf)

<https://stackoverflow.com/questions/3332947/when-is-it-practical-to-use-depth-first-search-dfs-vs-breadth-first-search-bf>

* If you know a solution is not far from the root of the tree, a breadth first search (BFS) might be better.
* If the tree is very deep and solutions are rare, depth first search (DFS) might take an extremely long time, but BFS could be faster.
* If the tree is very wide, a BFS might need too much memory, so it might be completely impractical.
* If solutions are frequent but located deep in the tree, BFS could be impractical.
* If the search tree is very deep you will need to restrict the search depth for depth first search (DFS), anyway (for example with iterative deepening).

|  |
| --- |
| **The following questions assess the criteria**: “*Detailed and effective application of set processes to solve simple and familiar problems”* |



SET THEORY – A set is an unordered collection of unique elements.

* A = {1, 2, 3}
* B = {3, 4, 5}

|  |  |  |  |
| --- | --- | --- | --- |
| Notation | Operation | Python | Result |
| A∩B | Intersection | print(A & B)  | { 3 } |
| A∪B | Union  | print(A | B)  | { 1, 2, 3, 4, 5 } |
| A-B | Difference (or relative complement) | print(A – B) | { 1, 2 } |
| B-A | print(B – A) | { 4, 5 } |
| A∆B | Symmetric difference | print(a ^ b)  | { 1, 2, 4, 5 } |

1. Desk check the following algorithm by recording the OUTPUT of A and B:

|  |  |  |
| --- | --- | --- |
| FUNCTION “Special” (parameters X, Y) IF X <= 3 THEN RETURN the product of X and 5 ELSE RETURN the product of X and YEND FUNCTIONBEGIN MAIN SET both A and B to 1 OUTPUT both A and B DO: INCREMENT A by 1 SET B to the result of “Special” with arguments A, B OUTPUT both A and B WHILE A < 5END MAIN | A | B |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

1. Calculate the resulting set from the following set binary operations:
* Let A = {1, 2, 3}
* Let B = {3, 4, 5}
* Let C = {1, 4, 5}

|  |  |
| --- | --- |
| Operations | Resulting Set (working out) |
| (A∆B)-C  |  |
| (A∪B) ∩ (B∪C) |  |
| ((C-A)-B)∪(A∩C) |  |
| If {2} in (C-A): ((A∪B)-(B∆A)) ∩ BElse: ((A∩B)∪(B∆A)) – BElse:  |  |
| BEGIN D = {} E = {} For x = 1 to 5 Append x to D Append D∪A to set E Next x if ( {5} in E ): Set E to D-B else: Set E to C∆AEnd | Set E only: |

*3. Draw the most efficient logic circuits for the following expressions. W, X, Y and Z are variables of type Boolean:*

1. IF NOT(X) IS TRUE
2. IF NOT( NOT ( X AND Y) ) IS TRUE
3. IF X OR Y IS TRUE
4. IF NOT (X OR Y) IS TRUE
5. IF (X OR Y) AND Z IS TRUE
6. IF (X AND Y) OR (NOT(X AND Z)) IS TRUE
7. IF W OR (X AND Y) OR NOT Z IS TRUE
8. IF NOT(W OR X) AND (Y OR Z) AND NOT ( NOT(Y)) IS TRUE

*4. Design complex logic gates for the following truth tables*:

|  |  |  |
| --- | --- | --- |
| x | y | Output |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

|  |  |  |
| --- | --- | --- |
| x | y | Output |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

|  |  |  |  |
| --- | --- | --- | --- |
| x | y | z | Output |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

|  |
| --- |
| **Criteria assessed implicitly throughout all questions in Part B**: “*comprehensive construction of documentation and fluent presentation of information using suitable communication conventions to convey meaning appropriate to the context*” (in presentation of technical ideas, design concepts, solutions and evaluations). |

**End of Part A: Knowledge and Application.**

**Part B - Managing an Exam Room: Case Study**

Due to small classes, three classes share the same exam room at the same time (as you are right now). At the moment the room is chaos. Timing and attendance is inaccurate, IPT and ITS students argue (as per usual), and all students sit wherever they want, so cheating is rife. The three teachers, students and Principal have all offered their **perspectives** on the current practice:

* Computer Science teacher: “I have two classes but unfortunately my IPT and ITS students can’t sit near each other, as they argue too much about which subject is better.”
* English teacher: “I’d like my students to sit together in a cluster, but in an alphabetical order, as we practice the alphabet every day in English. Also, every few desks (or so), I’d like to leave an empty desk, which I can leave a pile of student dictionaries on for easy access.”
* HPE teacher: “My students have exams that are half as long as the computer science exams. So they usually distract the other kids sitting near them when they pack up early, and also when they walk out of the room bouncing their basketballs. I don’t care where they sit, but the other teachers probably do.”
* Principal: “I don’t sit in the exam room, but since only one teacher has to be in there at a time supervising, I am sick of the three teachers fighting over who supervises the first third, second third or final third of the exam.”
* Students: “We’d like the ability to swap our seat with someone else if we were allowed to. Also it would be cool if we got told when there was 5 minutes left on the test, so we could guess answers to questions we haven’t answered yet.”



Most allocated exam rooms are set up in a square formation, with front and rear entrances for safety:

**Part B - Managing an Exam Room: Analysis**

Deconstruct the exam room setting, and interpret the feedback from multiple perspectives, to **analyze the required specifications** for an algorithm that will **solve the problems identified with managing an exam room**.

Your analysis could include:

* a list of suitable, salient **features** your algorithm will seek to deliver, with justification of the proposed features in regards to the problem(s) they will address;
* an overarching set of rules or logic that your algorithm must adhere to:
	+ rules or logic that will determine whether transactions are valid or invalid;
	+ required data inputs for your algorithm, and if necessary, explicitly state pre-conditions that you are going to either: specify limits for, assume defaults for, or presume will be met;
* a list of caveats, constraints or limits that will be outside the scope of your algorithm;
* a practical consideration of how your proposed algorithm could be implemented (e.g. via platform or system requirements, user hierarchy or access privileges, etc.)
* an analysis of which algorithmic concepts learnt in class could be utilized in developing a solution to the problem(s) addressed within this context.

|  |
| --- |
| **Criteria assessed**: “*Detailed interpretation and analysis of problems and situations from multiple perspectives*” |

Space below for diagrams if required, or room to keep writing if necessary:

**Part B - Managing an Exam Room: Synthesis**

Utilize appropriate algorithmic design methods and principles to implement your specified algorithmic solution for managing an exam room.

Your synthesis to this unrehearsed problem should be:

* significant in scope
* complex in nature
* make use of consistent algorithmic conventions

|  |
| --- |
| **Criteria assessed**: “*designed and developed effective solutions to unrehearsed or complex problems*” |

|  |
| --- |
|  |

|  |
| --- |
|  |

*Write on the back of this page if more room is needed.*

**Part B - Managing an Exam Room: Evaluation**

**Evaluate** your algorithm by applying self-determined and prescribed criteria, reasoning or evidence to draw conclusions and make future recommendations. Your evaluation should encompass a use of logic and reason in a range of evaluation approaches to achieve success.

|  |
| --- |
| **Criteria assessed**: “*… application of self-determined and prescribed criteria, reasoning and evidence to draw conclusions and make supported recommendations.* ” |

|  |  |
| --- | --- |
| **Prescribed Criteria** | **Definition** |
| Scalability | The ability of the algorithm to continue to function well when it (or its context) is changed in size or volume in order to meet a user need. |
| Efficiency | The requirement for the algorithm to perform its required tasks at the highest speed, or with the minimum resource usage possible. |
| Modularity | Separating the functionality of an algorithm into independent, interchangeable modules, to assist management, reusability and decrease clutter. |
| **Self-determined Criteria** | **Definition** |
|  |  |
|  |  |

Space below for diagrams if required, or room to keep writing if necessary: