Algorithms and Data Structures III (course 1DL481)
Uppsala University – Spring 2024
Report for Assignment n by Team t

Clara CLÄVER Whiz KIDD

23rd January 2024

This document imposes the *structure* of an assignment report for this course. The LaTeX source code of this document exemplifies almost everything you need to know about LaTeX in order to typeset a professional-looking assignment report (for this course). Use it as a starting point for imitation and delete everything irrelevant. The usage of LaTeX is *optional*, but highly recommended, for reasons that will soon become clear to those who have never used it before: any learning time is *outside* the budget of this course, but will hugely pay off, if not in this course then in the next course(s) you take and when writing a thesis or other scientific report.

Replace every blue text by your text (or just delete it, where appropriate, such as this and the preceding paragraphs) and drop the call to the `\todo` command, so that its argument appears in black.

Problem 1: Mixed Integer Programming (MIP)

Task a: Model. What are the variables, their meanings, their constraints, and the objective function? For example, for the investment design problem, one might write: "Let variable m_{ij} take value 1 if basket i invests in credit j, and value 0 otherwise, with $i \in 1\,..\,v$ and $j \in 1\,..\,b$. The constraint that each row must sum up to r can then be linearly modelled as

$$\forall i \in 1\,..\,v : \sum_{j \in 1..b} m_{ij} = r$$

Etc."

Task b: Implementation. Our model `servStatLoc.mod` is uploaded with this report (but not listed inside it): we checked that its constraints and objective function are linear (and we are aware that four points will otherwise be deducted from our score for this problem). We chose the MIP solver Gurobi for our experiments, which we ran on the NEOS server or under Linux Ubuntu 18.04 (64 bit) on an Intel Xeon E5520 of 2.27 GHz, with 4 processors of 4 cores each, with a 70 GB RAM and an 8 MB L3 cache (a ThinLinc computer of the IT department).

Task c: 10 Zones. The results are in Table 1.

Task d: 20 Zones. The results are in Table 1.

Task e: 40 Zones. The results are in Table 1.

z	s	v	c	time	objective value	optimality gap	brute-force
10	2	2	3	67.89	0.008740338682	0.00%	10^{17}
10	3	2	3				
10	4	2	3				
20	2	2	3	123.45	0.023246261350	0.00%	10^{23}
20	3	2	3				
20	4	2	3				
20	5	2	3				
20	6	2	3				
40	5	2	3				
80	8	2	3				
80	16	1	3				
120	10	2	3				
250	12	3	4	300.00		2.34%	n/a

Table 1: Service station location: runtime (in seconds), objective value, and optimality gap (in percent; positive if an optimal solution was not found and proven before timing out) using Gurobi, with a timeout of 300.00 CPU seconds. The right-most column gives the number of candidate solutions the brute-force search algorithm has to examine per second in order to match the runtime performance of Gurobi, if the instance was solved to proven optimality, and 'n/a' for 'non-applicable' otherwise. (The sample performance of this demo report is made up, but the two optimal objective values are correct!)

Task f: 80 Zones. The results are in Table 1.

Task g: 120 Zones. The results are in Table 1. Our model does not time out.

Task h: 250 Zones. The results are in Table 1. Our model times out, so our proposed algorithm for delivering a not necessarily optimal solution in reasonable running time is $\langle \dots \rangle$.

Task i: Brute-Force Algorithm. The size of the search space of a totally brute-force search algorithm is $\binom{z!}{\cos c} \cdot \log_s v$, because $\langle \dots \rangle$.

The numbers of candidate solutions this brute-force search algorithm has to examine per second in order to match the reported runtime performance of Gurobi on our model are given in the right-most column of Table 1, for each instance that Gurobi solved to proven optimality without timing out.

Problem 2: Stochastic Local Search (SLS)

The *investment design problem* is about finding a matrix of v rows and b columns of 0-1 integer values, such that each row sums up to r, with $v \geq 2$ and $b \geq r \geq 1$, and the largest dot product between all pairs of rows is minimised. Equivalently, one has to find v subsets of size r within a given set of b elements, such that the largest intersection of any two of the v sets has minimal size. An instance of the problem is parametrised by a triple $\langle v, b, r \rangle$.[1]

This is an abstract description of a problem that appears in finance: see Figure 1. In a typical investment design in finance, we have $4 \leq v \leq 25$ and $250 \leq b \leq 500$, with $r \approx 100$.

A lower bound on the number λ of shared elements of any pair among v subsets of size r drawn from a given set of b elements is given in [3]:

$$\mathrm{lb}(\lambda) = \left\lceil \frac{\left\lceil \frac{rv}{b} \right\rceil^2 ((rv) \bmod b) + \left\lfloor \frac{rv}{b} \right\rfloor^2 (b - ((rv) \bmod b)) - rv}{v(v-1)} \right\rceil \tag{1}$$

Task a: SLS Algorithm.

1. Representation. Describe how to represent the problem: what are the variables, their meanings, their constraints, and the objective function?
2. Initial Assignment. Describe an algorithm for generating (fast) a randomised initial assignment.
3. Move. Describe one or more moves that go from an assignment to a neighbouring assignment by changing the values of a few variables.
4. Constraints. Describe for each constraint how its satisfaction is either algorithmically checkable efficiently or guaranteed to be preserved by the previous two design choices.
5. Neighbourhood. Describe a neighbourhood based on the proposed moves. Derive a formula for computing the size of the neighbourhood in terms of the problem parameters. Discuss whether the neighbourhood makes the search space connected, in the sense that every feasible assignment (that is, every assignment satisfying all the constraints, whether optimal or not) is reachable from every initial assignment (you only need to sketch a proof if the search space is connected, and give a counterexample otherwise).

[1] Your report need not contain an explanation of the problem to be solved: you can assume the reader has read the problem statement in the assignment instructions. We just repeat it here for self-sufficiency of this document.

Figure 1: Wall Street (© www.forbes.com, 2014)

6. Cost Function. Describe a cost function, whose value is to be minimised during search.
7. Probing. Describe how a neighbouring assignment, as reachable by a move, can be probed efficiently: describe how the cost function can be evaluated efficiently and incrementally, and describe the data structures used to do so. Give, without proof, the time complexity of probing; ideally, it is (sub-)linear in the problem parameters.
8. Heuristic. Describe a heuristic for exploring (via probing) the neighbourhood and selecting a neighbouring assignment to commit to. State whether the neighbourhood is explored exhaustively and, if so, how you determine when it was exhausted. Explain how you ensure that the same neighbour is not probed twice during a given exploration.
9. Optimality. Describe how you use a bound on the objective value in order to terminate sometimes the search with proven optimality, as part of the heuristic.
10. Meta-Heuristic. Describe a meta-heuristic based on tabu search: explain how the tabu list is represented; choose (a formula for) its size; explain how fine-grained its content is; and describe how it can be looked up and maintained efficiently; note that the tabu list is not necessarily an actual list, but rather a concept; make sure that worsening moves are sometimes made.
11. Random Restarts. Describe how to detect or guess that a random restart should be made, as part of the meta-heuristic.
12. Optional Tweaks. Describe ideas that you used in order to improve your algorithm.

In summary, the local-search parameters (not the problem parameters v, b, r) are α and β.

Task b: Implementation. We chose the high-performance programming language Java, for which a compiler or interpreter is available on the Linux computers of the IT department. All source code is uploaded with this report (but not listed inside it). The compilation and running instructions are $\langle \dots \rangle$.

An executable called `InvDes` reads the problem parameters v, b, r as command-line arguments and writes to standard output a line with the space-separated values of v, b, r, the lower bound $\text{lb}(\lambda)$ on λ, and the achieved λ, followed by one line per row of a $v \times b$ matrix representing the solution, the 0-1 cell values being space-separated.

We validated the correctness of our implementation by checking its outputs on 1,258 instances via the provided polynomial-time solution checker.

Task c: Experiments. All experiments were run under Linux Ubuntu 18.04 (64 bit) on an Intel Xeon E5520 of 2.27 GHz, with 4 processors of 4 cores each, with a 70 GB RAM and an 8 MB L3 cache (a ThinLinc computer of the IT department).

We fine-tuned the local-search parameters as follows. Discuss the impact of the local-search parameters α and β on the performance of your SLS algorithm.

The median runtime (in seconds), median number of steps, and median achieved λ over 5 independent runs for each of the 21 instances of the assignment instructions are given in Table 2, for two configurations of values for the local-search parameters α and β. The timeout was 300.0 CPU seconds per run.[2]

[2]Hint: In order to save a lot of time, it is very important that you write a script that conducts the experiments for you and directly generates a result table (see the LaTeX source code of Table 2 for how to do that), which is then automatically imported, rather than manually copied, into your report: each time you change the code, it suffices to re-run that script and re-compile your report, without any tedious copying! The sharing of scripts is allowed and even encouraged.

				$\langle\alpha,\beta\rangle = \langle 10,5\rangle$			$\langle\alpha,\beta\rangle = \langle 20,8\rangle$			
v	b	r	lb(λ)	time	steps	λ	time	steps	λ	exact
10	30	9	2	300.0	123	3	45.0	678	2	10^{22}
12	44	11	2	300.0	901	3	234.5	5678	2	10^{21}
15	21	7	2	300.0	1023	4	300.0	6789	3	n/a
16	16	6	2	123.4	567	2	89.1	2345	2	10^{14}
9	36	12	3							
11	22	10	4							
19	19	9	4							
10	37	14	5							
8	28	14	6							
10	100	30	7							
6	50	25	10							
6	60	30	12							
11	150	50	14							
9	70	35	16							
10	350	100	22							
13	250	80	22							
10	325	100	24							
15	350	100	24							
9	300	100	25							
12	200	75	25							
10	360	120	32							

Table 2: Investment design: median runtime (in seconds), median number of steps, and median achieved λ, for two configurations of values for the local-search parameters α and β, over 5 independent runs per instance, with a timeout of 300.0 CPU seconds per run. The right-most column gives the number of candidate solutions the outlined exact algorithm has to examine per second in order to match the runtime performance of the seemingly best configuration of values for the local-search parameters, namely $\langle\alpha,\beta\rangle = \langle 20,8\rangle$, if the instance was solved to proven optimality, and 'n/a' for 'non-applicable' otherwise. (The sample performance of this demo report is made up!)

We observe that $\langle \dots \rangle$, because $\langle \dots \rangle$.

Task d: Exact Algorithm. An exact algorithm could work as follows: discuss its features (for instance, does it perform brute-force search?). The size of the search space of this exact algorithm is $\binom{r!}{\cos b} \cdot \log v$, because $\langle \dots \rangle$.

The number of candidate solutions this exact algorithm has to examine per second in order to match the runtime performance of the seemingly best configuration of values for the local-search parameters, according to Task c, of our stochastic local search algorithm is given in the right-most column of Table 2, for each instance solved to proven optimality. We think that $\langle \dots \rangle$, because $\langle \dots \rangle$.

Problem 3: Boolean Satisfiability (SAT)

Task a: Ordered Resolution. Consider the following formula in conjunctive normal form (CNF):

$$\varphi \equiv (x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge (x_5 \vee x_6) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_5) \\ \wedge (\neg x_3 \vee \neg x_5) \wedge (\neg x_2 \vee \neg x_4) \wedge (\neg x_2 \vee \neg x_6) \wedge (\neg x_4 \vee \neg x_6)$$

Perform ordered resolution on this formula, selecting the variables in the order given by their index (i.e., x_1 before x_2 before $\dots$). Show the result after each iteration. Based on your resolution, is φ satisfiable?

Task b: DPLL. Consider again the formula φ given in Task a. Explain in detail how the DPLL algorithm, when applied to φ, determines whether the formula is satisfiable. Assume that the variables are selected in the order given by their index (i.e., x_1 before x_2 before $\dots$), and that they are assigned 1 (i.e., True) before they are assigned 0 (i.e., False). Remember to perform unit propagation and to apply the pure-literal rule where possible, in order to prune parts of the search space.

Task c: CDCL. Consider the following CNF formula:

$$(x_1 \vee x_8 \vee \neg x_2) \wedge (x_1 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee x_4) \wedge (\neg x_4 \vee \neg x_5) \wedge (x_7 \vee \neg x_4 \vee \neg x_6) \wedge (x_5 \vee x_6)$$

Assume that x_7 has been assigned 0 at decision level 2, and that x_8 has been assigned 0 at decision level 3. Moreover, assume that the current decision assignment is $x_1 = 0$ at decision level 5. Draw a resulting implication graph (possibly on paper or a whiteboard, and include it using the syntax for Figure 1). Does the graph contain any conflicts? If so, then mark these clearly, and provide a conflict clause.

Task d: Encoding. Describe your encoding, citing either [2], or [1, Section 2.2.2], or both, if you use their ideas: first explain the meaning of the Boolean variables that you use in your formula $\varphi_{d,c,e}$; then explain the encodings by the help functions of the hint that you actually use (there is no need to explain $\textsc{AtMost}(k, x_1, \dots, x_n)$ if you use [2]); and finally explain how the constraints of the problem are encoded using those variables and help functions.

We chose the programming language $\langle \dots \rangle$, for which a compiler or interpreter is available on the Linux computers of the IT department. All source code is uploaded with this report (but not listed inside it). The compilation and running instructions are $\langle \dots \rangle$.

We validated the correctness and speed of our encoding by checking its outputs on 1,258 instances via the provided polynomial-time solution checker and by judging the runtime to be short.

Task e: Experiments. We chose the SAT solver MiniSat for our experiments. We used or did not use the provided script for running the experiments and tabling their results under Linux Ubuntu 18.04 (64 bit) on an Intel Xeon E5520 of 2.27 GHz, with 4 processors of 4 cores each, with a 70 GB RAM and an 8 MB L3 cache (a ThinLinc computer of the IT department).

The results are in Table 3. The trivially unsatisfiable instances (which are the ones that violate the inequality $e \leq \left\lfloor \frac{d-1}{c-1} \right\rfloor$) that were actually attempted in our experiments are $\langle 8, 2, 8 \rangle$, $\langle 10, 2, 10 \rangle$, $\langle 12, 2, 12 \rangle$, $\langle 14, 2, 14 \rangle$, $\langle 16, 2, 16 \rangle$, $\langle 15, 3, 8 \rangle$, $\langle \dots \rangle$, $\langle 16, 4, 6 \rangle$, $\langle \dots \rangle$, and $\langle \dots \rangle$. We observe that our encoding detects their trivial unsatisfiability in $\langle \dots \rangle$ time.

d	c	e	status	time	d	c	e	status	time	d	c	e	status	time
8	2	7	sat		12	3	4	sat		16	4	5	sat	
8	2	8	unsat		12	3	5	unsat		16	4	6	unsat	
10	2	9	sat		15	3	6	sat		20	4	4	sat	
10	2	10	unsat		15	3	7	sat		20	4	5	sat	
12	2	11	sat		15	3	8	unsat		20	4	6	?	
12	2	12	unsat		18	3	6	sat		24	4	4	sat	
14	2	12	sat		18	3	7	sat		24	4	5	sat	
14	2	13	sat		18	3	8	?		24	4	6	?	
14	2	14	unsat		21	3	6	sat		28	4	4	sat	
16	2	10	sat		21	3	7	sat		28	4	5	sat	
16	2	11	sat		21	3	8	sat		28	4	6	sat	
16	2	12	sat		21	3	9	?		28	4	7	?	
16	2	13	sat		24	3	6	sat		32	4	3	sat	
16	2	14	sat		24	3	...	sat		32	4	...	sat	
16	2	15	sat		24	3	9	sat		32	4	9	sat	
16	2	16	unsat		24	3	10	?		32	4	10	sat	

Table 3: Cruise design: satisfiability and runtime (in seconds) using MiniSat, with a timeout of 60.0 CPU seconds; a timeout is denoted by 't/o'; if no timeout occurred, then proven satisfiability is denoted by 'sat' and proven unsatisfiability by 'unsat', else trivial unsatisfiability is denoted by 'unsat' and the unknown status is denoted by '?'.

Problem 4: SAT Modulo Theories (SMT)

Task a: Encoding of instructions. We encode the transition between program states as follows for each MiniASM instruction:

- push_j:
  - In English: $\langle \ldots \rangle$.
  - In SMT syntax: $\langle \ldots \rangle$.
- pop:
  - In English: $\langle \ldots \rangle$.
  - In SMT syntax: $\langle \ldots \rangle$.
- dup:
  - In English: $\langle \ldots \rangle$.
  - In SMT syntax: $\langle \ldots \rangle$.
- plus:
  - In English: $\langle \ldots \rangle$.
  - In SMT syntax: $\langle \ldots \rangle$.
- neg:
  - In English: $\langle \ldots \rangle$.
  - In SMT syntax: $\langle \ldots \rangle$.
- read:
  - In English: $\langle \ldots \rangle$.
  - In SMT syntax: $\langle \ldots \rangle$.
- write:
  - In English: $\langle \ldots \rangle$.
  - In SMT syntax: $\langle \ldots \rangle$.

Task b: Partial-correctness checker. Describe your encoding. The final `assert` call, which ensures that the SMT solver produces `unsat` when needed, is $\langle \ldots \rangle$.

We chose the SMT solver Z3 for our experiments. We chose the programming language $\langle \ldots \rangle$, for which a compiler or interpreter is available on the Linux computers of the IT department. All source code is uploaded with this report (but not listed inside it). The compilation and running instructions are $\langle \ldots \rangle$.

Let swap be the abbreviation of push_0; write; push_1; write; push_0; read; push_1; read, that is changing the order of the two top-most numbers on the stack:

1. The program push_{10}; read is or is not reported by Z3 to be partially correct, because $\langle \ldots \rangle$.
2. The program push_1; dup; dup; write; read $\langle \ldots \rangle$

3. The program $\mathsf{push}_1; \mathsf{dup}; \mathsf{read}; \mathsf{dup}; \mathsf{neg}; \mathsf{plus}; \mathsf{plus}$ $\langle \ldots \rangle$

4. The program $\mathsf{push}_1; \mathsf{push}_0; \mathsf{read}; \mathsf{write}; \mathsf{push}_0; \mathsf{read}; \mathsf{read}$ $\langle \ldots \rangle$

5. The program $\mathsf{push}_{10}; \mathsf{push}_0; \mathsf{swap}$ $\langle \ldots \rangle$

6. The program $\mathsf{push}_{10}; \mathsf{dup}; \mathsf{read}; \mathsf{swap}; \mathsf{push}_1; \mathsf{plus}; \mathsf{read}; \mathsf{plus}$ $\langle \ldots \rangle$

7. The program $\mathsf{push}_{10}; \mathsf{dup}; \mathsf{push}_1; \mathsf{plus}; \mathsf{dup}; \mathsf{push}_1; \mathsf{plus}; \mathsf{plus}; \mathsf{plus}$ $\langle \ldots \rangle$

When Z3 produces `sat` for a partial-correctness check, the output of `get-model` means $\langle \ldots \rangle$.

Task c: Partial-equivalence checker. Describe your encoding. The final `assert` call, which ensures that the SMT solver produces `unsat` when needed, is $\langle \ldots \rangle$.

We chose the SMT solver Z3 for our experiments. We chose the programming language $\langle \ldots \rangle$, for which a compiler or interpreter is available on the Linux computers of the IT department. All source code is uploaded with this report (but not listed inside it). The compilation and running instructions are $\langle \ldots \rangle$.

Assuming that variable x is stored at heap address 0 and variable y is stored at heap address 1, the program $t := x;\ x := y;\ y := t$, translated into

$$\mathsf{push}_0; \mathsf{read}; \mathsf{push}_1; \mathsf{read}; \mathsf{push}_0; \mathsf{write}; \mathsf{push}_1; \mathsf{write}$$

and the program $x := x + y;\ y := x - y;\ x := x - y$, translated into

$$\mathsf{push}_0; \mathsf{read}; \mathsf{push}_1; \mathsf{read}; \mathsf{plus}; \mathsf{dup}; \mathsf{push}_1; \mathsf{read}; \mathsf{neg}; \mathsf{plus}; \mathsf{dup}; \mathsf{push}_1; \mathsf{write}; \mathsf{neg}; \mathsf{plus}; \mathsf{push}_0; \mathsf{write}$$

are or are not reported by Z3 to be partially equivalent, because $\langle \ldots \rangle$. When Z3 produces `sat` for a partial-equivalence check, the output of `get-model` means $\langle \ldots \rangle$.

Task d: Extended language. We encode the transition between program states for the $\mathsf{cmp}_\circ$ instruction as follows:

- In English: $\langle \ldots \rangle$.
- In SMT syntax: $\langle \ldots \rangle$.

For encoding the jmp_j instruction (where $j \geq 0$), we propose $\langle \ldots \rangle$. The difficulty of encoding one or more jmp_j instructions lies in $\langle \ldots \rangle$. With our approach for encoding jmp_j, the partial correctness of programs is determined by $\langle \ldots \rangle$.

Feedback to the Teachers

Write a paragraph, which will not be graded, describing your experience with this assignment. You may also do so anonymously, by whichever channel you choose. Which tasks were too difficult or too easy? Which tasks were interesting or boring? (Recall that experiments are inevitable.) This will help us improve the course for the coming years.

References

[1] S. Prestwich. CNF encodings. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 2, pages 75–97. IOS Press, 2009. Available at `https://dx.doi.org/10.3233/978-1-58603-929-5-75` and `https://www.researchgate.net/publication/242029085_CNF_encodings`.

[2] C. Sinz. Towards an optimal CNF encoding of Boolean cardinality constraints. In P. van Beek, editor, *CP 2005*, volume 3709 of *LNCS*, pages 827–831. Springer, 2005. Available at `https://dx.doi.org/10.1007/11564751_73`; extended and corrected ($\mathrm{LT}_{\mathrm{SEQ}}^{n,k}$ has $2nk+n-3k-1$ clauses) at `http://www.carstensinz.de/techreports/CardConstraints.pdf`.

[3] O. Sivertsson, P. Flener, and J. Pearson. A bound on the overlap of same-sized sets. *Annals of Combinatorics*, 12(3):347–352, October 2008. Available at `https://dx.doi.org/10.1007/s00026-008-0355-0`.

More LaTeX and Technical Writing Advice

Unnumbered itemisation (only to be used when the order of the items does *not* matter):[3]

- Unnumbered displayed formula:
$$E = m \cdot c^2$$
- Numbered displayed formula, which is cross-referenced somewhere:
$$E = m \cdot c^2 \tag{2}$$
- Formula — the same as formula (2) — spanning more than one line:
$$\begin{gathered} E \\ = m \cdot c^2 \end{gathered}$$

Numbered itemisation (only to be used when the order of the items *does* matter):

1. First do this.
2. Then do that.
3. If we are not finished, then go back to Step 2, else stop.

Tables and elementary mathematics are typeset as exemplified in Table 4; see `http://tug.ctan.org/info/short-math-guide/short-math-guide.pdf` for many more details.

Use `\mathit{...}` in mathematical mode for each multiple-letter identifier in order to avoid typesetting the identifier like the product of single-letter ones. For example, note the typographic difference between the identifier WL, obtained through `$\mathit{WL}$`, and the product WL, where there is a small space between the W and the L, obtained through `$WL$`.

[3]Use footnotes very sparingly, and note that footnote pointers are *never* preceded by a space and always glued immediately *behind* the punctuation, if there is any.

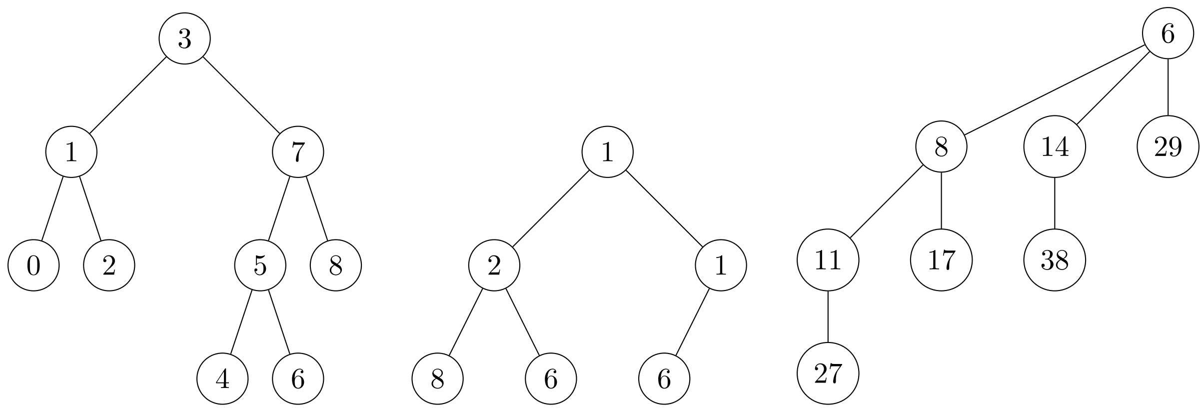

Figure 2: A binary search tree (on the left), a binary min-heap (in the middle), and a binomial tree of rank 3 (on the right).

Do *not* use programming-language-style lower-ASCII notation (such as ! for negation, && for conjunction, || for disjunction, and the equality sign = for assignment) in algorithms or formulas (but rather use $\neg$ or **not**, $\wedge$ or & or **and**, $\vee$ or **or**, and $\leftarrow$ or $:=$, respectively), as this testifies to a very strong confusion of concepts.

Figures can be imported with `\includegraphics` or drawn inside the LaTeX source code using the highly declarative notation of the `tikz` package: see Figure 2 for sample drawings. It is perfectly acceptable in this course to include scans or photos of drawings that were carefully done by hand.

If you are not sure whether you will stick to your current choice of notation or terminology, then introduce a new (possibly parametric) command. For example, upon

```
\newcommand{\Cardinality}[1]{\left\lvert#1\right\rvert}
```

the formula `$\Cardinality{S}$` typesets the cardinality of set S as $|S|$ with autosized vertical bars and proper spacing, but upon changing the definition of that parametric command to

```
\newcommand{\Cardinality}[1]{\# #1}
```

and recompiling, the formula `$\Cardinality{S}$` typesets the cardinality of set S as $\# S$. You can thus obtain an arbitrary number of changes in the document with a *constant*-time change in its source code, rather than having to perform a *linear*-time find-and-replace operation within the source code, which is painstaking and error-prone. The source code of this document has some useful predefined commands about mathematics and algorithms.

Use commands on positioning (such as `\hspace`, `\vspace`, and `\noindent`) and appearance (such as `\small` for reducing the font size, and `\textit` for italics) very sparingly, and ideally only in (parametric) commands, as the very idea of mark-up languages such as LaTeX is to let the class designer (usually a trained professional typesetter) decide on where things appear and how they look. For example, `\emph` (for emphasis) compiles (outside italicised environments, such as `theorem`) into *italics* under the `article` class used for this document, but it may compile into **boldface** under some other class.

If you do not (need to) worry about *how* things look,
then you can fully focus on *what* you are trying to express!

Note that *no* absolute numbers are used in the LaTeX source code for any of the references inside this document. For ease of maintenance, `\label` is used for giving a label to something that is automatically numbered (such as an algorithm, equation, figure, footnote, item, line, part, section, subsection, or table), and `\ref` is used for referring to a label. An item in the bibliography file is referred to by `\cite` instead. Upon changing the text, it suffices to recompile, once or twice, and possibly to run BibTeX again, in order to update all references consistently.

Always write `Table~\ref{tab:maths}` instead of `Table \ref{tab:maths}`, by using the non-breaking space (which is typeset as the tilde ∼) instead of the normal space, because this avoids that a cross-reference is spread across a line break, as for example in "Table 4", which is considered poor typesetting.

The rules of English for how many spaces to use before and after various symbols are given in Table 5. Beware that they may be very different from the rules in your native language.

☞ Feel free to report to the head teacher any other features that you would have liked to see discussed and exemplified in this template document.

Topic	LaTeX code	Appearance
Greek letter	`$\Theta,\Omega,\epsilon$`	Θ,Ω,ϵ
multiplication	`$m \cdot n$`	$m \cdot n$
division	`$\frac{m}{n}, m \div n$`	$\frac{m}{n}, m \div n$
rounding down	`$\left\lfloor n \right\rfloor$`	$\left\lfloor n \right\rfloor$
rounding up	`$\left\lceil n \right\rceil$`	$\left\lceil n \right\rceil$
binary modulus	`$m \bmod n$`	$m \bmod n$
unary modulus	`$m \equiv n \mod \ell$`	$m \equiv n \mod \ell$
root	`$\sqrt{n},\sqrt[3]{n}$`	$\sqrt{n},\sqrt[3]{n}$
exponentiation, superscript	`$n^{i}$`	n^{i}
subscript	`$n_{i}$`	n_{i}
overline	`$\overline{n}$`	$\overline{n}$
base 2 logarithm	`$\lg n$`	$\lg n$
base b logarithm	`$\log_b n$`	$\log_b n$
binomial	`$\binom{n}{k}$`	$\binom{n}{k}$
sum	`\[\sum_{i=1}^n i\]`	$\displaystyle\sum_{i=1}^n i$
numeric comparison	`$\leq,<,=,\neq,>,\geq$`	$\leq,<,=,\neq,>,\geq$
non-numeric comparison	`$\prec,\nprec,\preceq,\succeq$`	$\prec,\nprec,\preceq,\succeq$
extremum	`$\min,\max,+\infty,\bot,\top$`	$\min,\max,+\infty,\bot,\top$
function	`$f\colon A\to B,\circ,\mapsto$`	$f\colon A\to B,\circ,\mapsto$
sequence, tuple	`$\langle a,b,c \rangle$`	$\langle a,b,c \rangle$
set	`$\{a,b,c\},\emptyset,\mathbb{N}$`	$\{a,b,c\},\emptyset,\mathbb{N}$
set membership	`$\in,\not\in$`	$\in,\not\in$
set comprehension	`$\{i \mid 1 \leq i \leq n\}$`	$\{i \mid 1 \leq i \leq n\}$
set operation	`$\cup,\cap,\setminus,\times$`	$\cup,\cap,\setminus,\times$
set comparison	`$\subset,\subseteq,\not\supset$`	$\subset,\subseteq,\not\supset$
logic quantifier	`$\forall,\exists,\nexists$`	$\forall,\exists,\nexists$
logic connective	`$\land,\lor,\neg,\Rightarrow$`	$\land,\lor,\neg,\Rightarrow$
logic	`$\models,\equiv,\vdash$`	$\models,\equiv,\vdash$
miscellaneous	`$\&,\#,\approx,\sim,\ell$`	$\&,\#,\approx,\sim,\ell$
dots	`$\ldots,\cdots,\vdots,\ddots$`	$\ldots,\cdots,\vdots,\ddots$
dots (context-sensitive)	`$1,\dots,n; 1+\dots+n$`	$1,\dots,n; 1+\dots+n$
parentheses (autosizing)	`$\left(m^{n^k}\right),(m^{n^k})$`	$\left(m^{n^k}\right),(m^{n^k})$
identifier of >1 character	`$\mathit{identifier}$`	$\mathit{identifier}$
hyphen, n-dash, m-dash, minus	`-, --, ---, $-$`	-, –, —, $-$

Table 4: The typesetting of elementary mathematics. Note very carefully when italics are used by LaTeX and when not, as well as all the horizontal and vertical spacing performed by LaTeX.

		number of spaces after	
		0	1
number of spaces before	0	/ -	, : ; . ! ?)] } ’ ” %
	1	([{ ‘ “	– (n-dash) — (m-dash)

Table 5: Spacing rules of English