


Algorithmics	Student information	Date	Number of session
	UO: 270534	5/4/2021	9
	Surname: Iglesias Préstamo	 Escuela de Ingeniería Informática Universidad de Oviedo	
	Name: Ángel		



Activity 1. [Implementation]

- Task 1 – Brief explanation of the designed heuristics.** Basically, the heuristic that I have designed consists of performing a distinction: if it prunes or not. In such a case, we discard the branch we were developing. The pruning condition is very straight forward: when we have inserted so many songs to both: block A and block B, that we surpassed the (previously established) threshold. In any other case we calculate the heuristic value depending on the expected score the collection can reach. This expected score is calculated by “trying” to insert the remaining songs in the list in both blocks until we can no longer insert in any of them, and comparing which branch has the highest chance of being the path to the solution.
- Task 2 – Measurement of the times for different problem sizes, compared to the Backtracking technique.**

<i>n</i>	<i>Branch and Bound (ms)</i>	<i>Backtracking</i>
25	851	290
50	3,095	124
100	20,563	120
200	271,431	103
400	Took so much	127

This test is not as good as it should, the times may vary a lot in case they find the solution in some best-case or worst-case. Not only that, but also, my computer is not as good as it could, and it takes ages to compute some calculations. However, the backtracking solution is better. As when it finds the solution, we have finished executing. Probably these results could be wrong, however my implementations work just fine. The reason to this is explained furthermore in the following section.

Activity 2. [Discussion]

1. **Task 2 – Discussion about the efficiency of both techniques.** I think both techniques are valid; however, Branch and Bound implementation could be better as we do not necessarily need to develop the whole tree, in case we trim some nodes, as can be seen when running the `BestListBranchAndBoundTimes.java` class. As far as we know, it depends on the list, and in such class, we generate them randomly; having that said, for example:

- a. Processed Nodes: 893
- b. Generated Nodes: 961
- c. Trimmed Nodes: 65

As you can see, 65 nodes are stopped: preventing them from being developed. This means, some sort of optimization. This is caused by the fact that when BNB algorithms realize there is a better solution, they abandon the “wrong ones”. Whereas the Backtracking technique finds solutions until it reaches a valid one (or all in case the execution is not successfully stopped). I believe backtracking is a good general-purpose approach for solving problems; meanwhile BNB solves the optimization problems – those that have a bounding function. However, the implementation shares some aspects such as the expand part of the code. The bad part of this kind of algorithms is that – in worst-cases – they require exponential complexities, even if in cases at which they (luckily) reach a solution in the first try, can be reasonable efficient. With that said, in my view backtracking is more efficient, as it finds the solution for the subproblems and keeps dividing and solving procedurally, while BNB only finds the solution at the end – when all the solution space has been explored.