

University of Montenegro
Faculty of natural sciences
Major: Computer Science



Creating an npm library for priority order

Mentor:

Prof. Dr. Aleksandar Popovic

The student:

Dejan Todorovic, 4/18

Podgorica, 2022

The content

1. Introduction.....
3 2 Technologies and tools
5 2.1 JavaScript	5
2.2 TypeScript	5 2.3
Node.js	6 2.4
Node package manager	6 2.5
Travis	6 2.6
Docker	7 3 Priority
order	8 3.1
Heap	8 4 Steps in
building a library.	11 4.1
Preparation	11 4.2
Installing the node	11 4.3
Initialization bi libraries	11 4.4
Initializing git and publishing on github	12 4.5 Installing
typescript	14 4.6 Formatting and
lining code	15 4.7 Creating interfaces and
types	17 4.8 Implementation of
functions.	18 4.9 Building a
library	25 4.10 Adding a LICENSE
file	30 4.11 Creating
examples	31 4.11.1 Creating
examples in JavaScript	31 4.11.2 Creating
examples in TypeScript	35 4.12 Test Writing
and Test Coverage	37 4.13 Writing a README
file	47 4.14 Using Travis to run tests
automatically	54 4.15 Publishing the
library	55 5
Conclusion	58 6
Bibliography ...	59

1. Introduction

JavaScript as a language used by search engines has long been very popular, and it is joined by Node.js, whose popularity has grown significantly in recent years due to its extremely low weight and great flexibility. Node.js and JavaScript are used by millions of developers around the world. When creating projects based on these technologies, we use various packages that facilitate development.

These packages are managed by Node Package Manager, and are characterized by being developed by both large companies and lay people. There are more than one million and eight hundred thousand packages on the npm register.

The work arose from the need of the author for a library that offers the use of priority order, such that the library user can submit a comparator that will be used when comparing elements. At the time of the beginning of the work, a search of npm registers showed that such a library does not exist, so the author decided to develop it himself.

The developed library is called priority-queue-with-custom-comparator and can be found on the npmjs website (<https://www.npmjs.com/package/priority-queue-with-custom-comparator>), but also in other packages managers who use the npm registry, under the same name.

The library provides support for both JavaScript and TypeScript, has application examples for both as well as example projects, written tests for all public functions, documentation of the library but also all public functions, written npm scripts, and integration with Travis which automatically performs tests when pushing code on GitHub.

It is written in TypeScript, for easier development and maintenance, and so that there would be no need for separate subsequent writing of types for users of this library, who will use it with TypeScript.

The code is open-source, and lining¹ and formatting are used so that anyone who wants to help further the development of the library can do so without violating the code format, as well as some eslint² rules.

Example projects contain Dockerfiles that can be used to create a docker image with all the tools needed to run them.

In the background lies a heap implemented over the string. In this paper, the heap is approached as a data structure, but the topic of this paper is not to present evidence of its correctness, nor to prove the temporal / spatial complexity of operations.

This library is published on GitHub, at:

<https://github.com/dejtor/priority-queue-with-custom-comparator>

¹ Lining is an automatic source code check for program and style errors. This is done using the tools for lining (otherwise known as linter).

² ESLint is a static code analysis tool (code analysis without executing it) to identify problematic forms contained in JavaScript code. It was created by Nicholas C. Zakas in 2013.

This paper can also serve as a guide for anyone who wants to create their own library and publish it on the npm register.

In addition to the chapter Introduction and Conclusion and bibliography, the paper contains 3 more chapters.

The second chapter, "Technologies and Tools", briefly describes the technologies and tools used to create the paper.

The third chapter, "Priority Order", presents an abstract data type that is implemented in many languages, the priority order. The heap is also described, because it is in the background of the implemented priority order.

The fourth chapter, "Steps in Creating a Library", describes in detail all the steps the author took to create a library.

2 Technologies and tools

2.1 JavaScript JavaScript

(JS) is an interpretive, object-oriented language with functions as a central concept.

It is best known as a scripting language for web pages, but is also used in many non-browser environments.

Supports object-oriented, imperative and functional programming styles.

In short, JavaScript is a dynamic scripting language that supports the construction of prototype-based objects.

The basic syntax is intentionally similar to Java and C++ in order to reduce the number of new concepts needed for language learning. Language constructs, such as if statements, for and while loops, and switch and try-catch blocks work the same as in these languages (or so).

It is characterized by the fact that it is actually defined as a standard, ECMAScript standard, and it is up to search engines and other tools to implement the functionality. So the most popular are Google's V8 engine (which uses Google Chrome, newer versions are searchable Opera, Node.js), JavaScriptCore (Safari), Chakra (Internet Explorer), but there are other implementations.

2.2 TypeScript TypeScript

is a programming language developed and maintained by Microsoft. It is a strict syntactic superset of JavaScript and adds the optional static addition of types to the language. It's free and open-source.

It is designed for large application development and translation into JavaScript. All existing JavaScript programs are also valid TypeScript programs. It can be used on both the server and client side. Well-known front-end frameworks involve its use, while Node.js supports TypeScript in addition to standard JavaScript.

TypeScript supports definition files that can contain information about the type of existing JavaScript libraries, much like C++ header files can describe the structure of existing object files.

This allows other programs to use the values defined in the files as if they were statically typed TypeScript entities. There are third-party header files for popular libraries such as jQuery, MongoDB, and D3.js. TypeScript headers for basic Node.js modules are also available, allowing Node.js to be developed within TypeScript.

The TypeScript compiler itself is written in TypeScript and translated into JavaScript. It is licensed under the Apache 2.0 license. TypeScript is included as a programming language in Microsoft Visual Studio 2013 Update 2 and later, in addition to C# and other Microsoft languages.

TypeScript developers were looking for a solution that would not compromise compatibility with the standard and its support for multiple platforms. Knowing that the current ECMAScript standard promises support for class-based programming, TypeScript is based on that proposal. This led to a JavaScript compiler with a set of syntactic language extensions, a template-based superset that converts extensions to plain JavaScript.

2.3 Node.js

Node.js is an open-source, cross-platform, back-end JavaScript runtime system that runs on the V8 engine and executes JavaScript code outside the web browser. Node.js allows developers to use JavaScript to write command-line tools and server-side scripting. Node.js represents the "JavaScript everywhere" paradigm, bringing together web application development around a single programming language, rather than different server-side and client-side scripting languages.

Node.js has an event management architecture capable of asynchronous I / O. These design choices aim to optimize bandwidth and scalability in web applications with many input / output operations, as well as for real-time web applications (e.g. real-time communication programs and browser games).

2.4 Node package manager

npm is the default package manager for the JavaScript runtime environment Node.js. It consists of a command line client, also called npm, and a network database of public and paid private packages, called the npm registry. The registry is accessed through the client, and the available packages can also be searched through the npm website. The package manager and registry are managed by npm, Inc. npm is included as a recommended feature in the Node.js installer. The registry does not have any verification procedure for submission, which means that packages found there can be of poor quality, insecure or malicious. Instead, npm relies on user reports to remove packages if they violate guidelines by being of poor quality, insecure, or malicious. npm presents statistics including the number of downloads and the number of dependent packages to help developers assess the quality of packages.

There are a number of open source alternatives to npm for installing modular JavaScript, including `ied`, `pnpm`, `npm`, `Yarn`, and `Yarn 2 (Berry)`. They are all compatible with the public npm registry (but Berry is not one hundred percent) and use it by default, but provide different client-side experiences, usually focused on improving performance and determinism compared to the npm client.

2.5 Travis

Travis CI is a hosted continuous integration service used to create and test software projects hosted on GitHub and Bitbucket.

Travis CI is the first CI service to provide free open source projects.

Continuous integration is the practice of merging small code changes often - instead of merging large changes at the end of the development cycle. The goal is to build healthier software by developing and testing in smaller steps.

As a platform for continuous integration, Travis CI supports the development process by automatically testing code changes, providing instant feedback on the success of the change.

2.6 Docker

Docker is a set of platform-as-a-service (PaaS) products that use OS-level virtualization to deliver software in packages called containers. The containers are isolated from each other and connect their own software, libraries and configuration files. They can communicate with each other through well-defined channels. Because all containers share the services of a single operating system core, they use fewer resources than virtual machines.

Containers run from the image (s) for Docker, which we define in Dockerfiles.

The service has both free and premium levels. The software that hosts the containers is called the Docker Engine. It was first launched in 2013 and is being developed by Docker, Inc.

3 Priority order

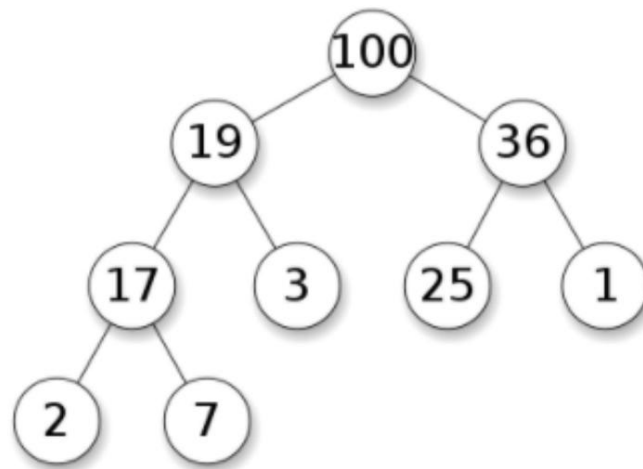
In computing, an associated row is an abstract data type, which is similar to a regular row or stack, but which additionally has an associated priority for each element. In order of priority, the element with the highest priority is taken before the element with the lower priority. If two elements have the same priority, then they are taken based on its position in the list.

Priority order is mostly implemented through the hip, but it is conceptually different from it. Priority is an abstract concept like "list" or "map". Just as a list can be implemented as a linked list or as an array, a priority row can be implemented over a hip or through other methods such as an unordered array.

3.1 Heap

In computer science, a stack is a data structure organized on the principle of a tree that must satisfy the condition of a stack: the key of each node is greater than or equal to the keys of their sons. When the maximum number of children of one node is two, it is a binary heap.

Prikaz drveta



Prikaz niza

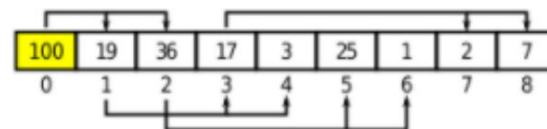


Figure 1 - Bunch

The bunch is extremely important in some algorithms, such as Dijkstra's algorithm and heapsort. A bunch is a very useful data structure when a node of the highest or lowest priority needs to be removed.

A node is one memory unit and contains a key, ie. data. Each node has its ancestor, except the root of the tree which represents the top of the hierarchy. If node A is the ancestor of node B, then B is a descendant of A. A node without descendants is called a leaf, and a node that is not a leaf is called an internal node. The key to each node is always greater than or equal to the keys of its descendants, and the biggest key is at the root of the tree. There is no special connection between nodes on the same level or brothers. As the heap is a binary tree, it has the smallest possible height - $O(\log n)$ for a heap with n nodes.

The element is added to the pile by making it the leaf, the rightmost on the last level, that is, such that it occupies the last position in the row. It then rises to the root, as long as the value of that node is greater than the value of its parent, their positions are reversed. This operation is performed in logarithmic time.

The removal of the highest priority element (root) is done by replacing the place with the rightmost leaf, and then the new root is lowered. As long as the value of the root is less than the value

one of his sons, he is replaced by a more valuable son. This operation is also performed in logarithmic time.

When creating a stack of strings, an algorithm can be used as when adding elements to the stack. For each element that is not a leaf, the process of its descent through the tree is started, as long as it is less of a priority than one of its sons. This procedure creates a pile in linear time.

4 Steps in building a library

4.1 Preparation To

begin with, it is necessary to choose the name of the library. The names of all libraries in the npm registry must be unique, so it is enough to check on the npm website (<https://www.npmjs.com/>) whether there is already a library with the same name. Prior to the creation of this project, there was no library with the name `priority-queue-with-custom-comparator` on the npm registry, so the author chose that name.

4.2 Installing the node

The recommendation of the authors of this paper is to download the latest stable version (LTS) of Node.js. Node can be installed independently or via `nvm` (Node Version Manager), a tool that offers easier switching from one version of the node to another, and thus easier development.

The installer will install Node.js runtime, npm package manager, documentation shortcuts, and register Node.js and npm in the path by default, so that they can be called from cmd. It is not necessary to install tools for native modules.

At the time of writing, the current version of the node was 14.

4.3 Library Initialization

It was necessary to create a folder that would be called the same as the library. Run cmd or PowerShell inside the folder and start the initialization process with the command `"npm init -y"`. npm then generates a `package.json` file.

```
{
  "name": "priority-queue-with-custom-comparator",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\ \"Error: no test specified \\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Table 1 - initial package.json

The package.json file is the heart of every Node project, it contains information about the project, dependencies on other packages, scripts, etc ..

A minor modification of this file describes this project more appropriately.

```
{
  "name": "priority-queue-with-custom-comparator",
  "version": "0.0.1",
  "description": "Priority queue data structure where you are able to set your own compare function.",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "priority-queue"
  ],
  "author": "Dejan Todorovic",
  "license": "ISC"
}
```

Table 2 - package.json after modification

4.4 Initializing git and publishing on github

Git is a version control tool. Version control tools are used on every serious project, both to track the history of changes, and to make it easier to restore the code and make it easier for more people to work on the same project.

GitHub is an online hosting service for git.

The git installer can be downloaded from <https://git-scm.com/downloads>.

Then it was necessary to create an account on GitHub or log in to the existing one, at <https://github.com/>.

In the next step, it was necessary to create a repository on GitHub. A repository is a collection of files, settings, and history associated with a single project.

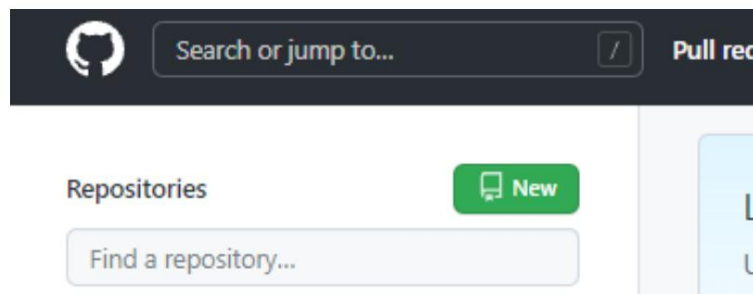


Figure 2 - part of the home page on GitHub


On the home page there is a link to create a new repository. Clicking the "New" button takes the user to the page to create a new repository.

Create a new repository


A repository contains all project files, including the revision history. Already have a repository elsewhere?

Owner *

Repository name *


 dejtor

/


priority-queue-with-custom-comparator 

Great repository names are short and memorable. Need inspiration? How about [ul](#)

Description (optional)

☒  Public

Anyone on the internet can see this repository. You choose who can commit.

☐  Private

You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

☐ Add a README file

This is where you can write a long description for your project. [Learn more.](#)

☐ Add .gitignore

Choose which files not to track from a list of templates. [Learn more.](#)

☐ Choose a license

A license tells others what they can and can't do with your code. [Learn more.](#)

Create repository

Figure 3 - creating a repository

The repository was given the same name as our project, chosen to be public (to make the library open-source) and to create a repository by clicking "Create repository".

```
git init
git commit -m "first commit" git
branch -M master git remote
add origin https://github.com/dejtor/priority-queue-with-custom-comparator.git git push -u origin
master
```

Table 3 - Git commands

It was necessary to execute the commands from Table 3 at the location where our project is located. These commands will initialize the git repository, put all the files in the first commit, which will go to the master branch, put the remote address from github and push commit to the repository.

Commit represents the smallest unit, ie the smallest change in the code that is made. Each commit should represent one whole. It is good practice that after it the project can be started normally, ie that there are no changes that will "break" the project.

Commitments can be placed on different branches for the sake of the organization, and thus separate the development of different functionalities. The branches that were created for the development of various functionalities, at the end of the development of that functionality, are merged into the main branch, which contains a stable project with all the fully developed functionalities. If only one person is working on one simple project, he can use the main branch for everything. The main branch is mostly called the master, but due to political correctness, the possibility of renaming it has been introduced recently.

Remote is the address where the online repository is hosted.

Pushing or pushing code to a branch is sending a certain number of commitments from our local repository to the hosted repository.

The recommendation of the authors of this paper is to use the integrated code editor tool or some auxiliary tool for working with git instead of cmd to make it easier to work with git.

Certain files do not want to be put on git, and this is emphasized in the .gitignore file that is created in the project's home folder.

```
node_modules
lib coverage
sample-projects
\ ** \ node_modules sample-projects \ **
\ dist
```

Table 4 - .gitignore

In gitignore, it is emphasized that we do not want to keep the node_modules folder on git, which contains installations of all our dependencies, the lib folder, which contains transpiled files, and the coverage folder, which contains data on project coverage by tests.

In the continuation of the work, the author will not pay attention to which changes are made in which commit and when the code should be pushed to a branch.

4.5 Installing typescript

```
npm install --save-dev typescript
```

Table 5 - TypeScript installation command

To install the typescript, it was enough to do the command in Table 5. - The save-dev flag indicates that this package will be used only on development environments. Also, the package can be installed globally using the -g flag, if typescript is planned to be used in other projects as well.

Both commands can be executed, so the version installed for this project will be used for this project. After installation, one can find typescript as devDependency (dependency necessary for

project development) in the package.json file, and package-lock.json will be created, a file that contains versions of its dependencies for each package.

```
{
  "compilerOptions":
    {"target": "es2017",
     "module": "commonjs",
     "declaration": true, "outDir":
     "./lib", "esModuleInterop":
     true
    }
}
```

Table 6 - tsconfig.json

```
{
  "extends": "./tsconfig.json",
  "include": ["src"], "exclude":
    ["node_modules", "** / __
    tests __ / **", "lib"
  ]
}
```

Table 7- tsconfig.build.json Both

tsconfig.json and tsconfig.build.json files have been created. Tsconfig.build.json describes the options required to compile a project. tsconfig.json will be its superset and will contain the configuration necessary for tests and code lining.

A version of ECMAScript es2017 was used, and the modules were compiled according to the commonjs (Node.js style module implementation) standard recommended by the official typescript documentation. declaration setting is set to true, so that when translating into JavaScript code, you also get a file with definitions.

4.6 Formatting and lining code

It is very important that the code follows some standards, especially if we want to work on it as a team. To achieve this, we can partially help by formatting and lining the code.

Code lining is a static analysis of code, ie analysis of code without its execution, and monitors not only whether the code is written in the expected format, but also whether we adhere to all rules.

Prettier was used for formatting the code, and eslint, javascript linter was used for linking.

```
npm install --save-dev @typescript-eslint / eslint-plugin @typescript-eslint / parser eslint eslint-config
prettier eslint-plugin-prettier prettier
```

Table 8 - commands for installing linters and formats

They are installed together with all the necessary packages, and then the necessary configuration files are created.

```
{"singleQuote": true,
  "trailingComma": "all"}
```

Table 9 - .prettierrc

The .prettierrc file was used to configure Prettier, where it is noted that the last element of the array in the list is followed by a comma, if the elements are separated by a new row. It is also noted that single quotes are used instead of double quotes wherever possible.

```
module.exports =
{parser: '@typescript-eslint /
parser', parserOptions: {project:
'tsconfig.json', sourceType:
'module'}, plugins: ['@typescript-
eslint / eslint-plugin'], extends:
['plugin: @typescript-eslint / recommended',
'plugin: prettier / recommended'], root: true,
env: {node: true, is: true}, ignorePatterns:
['.eslintrc.js'], rules: {'@typescript-eslint /
interface-name-prefix': 'off', '@typescript-
eslint / explicit-function-return-type': 'off', '@
typescript-eslint / explicit-module-boundary-
types': 'off', '@typescript-eslint / no-explicit-
any': 'off'},};
```

Table 10 - .eslintrc.js

The .eslintrc.js file was used to configure eslint. It emphasizes that it uses the default settings, and that it also looks at Prettier's rules. If we want to add other eslint rules, we can

to do so by adding a rules option, and passing an object that contains additional eslint rules. This project adds rules for naming interfaces, emphasizing which type returns a function, but also which type are arguments for public functions, and using the type any.

It was still necessary to write scripts for lining and formatting, and add them to package.json.

```
...
"scripts": {
  ...
  "format": "prettier --write \" src / ** / *. ts \" \" __ tests __ / *. ts \" \" sample-projects / ** / *. [tj] s \" \" , \" lint
  \": \" eslint \" {src, sample-projects, __ tests __} / ** / *. [tj] s \" --fix \" ,
  ...
},
...
```

Table 11 - scripts for formatting and lining in package.json

The Prettier script determines in which directories we format files, and notes Prettier that he is free to correct the file himself wherever possible. The eslint script works similarly.

Running defined scripts is very easy.

```
npm run script_name
```

Table 12 - Running the script

It was later set up for these scripts to run on their own during certain events.

4.7 Creating Interfaces and Types

Due to the neatness of the code and the possibility of using types for TypeScript, the interfaces and types, and the implementation of this library, are separated. So in the src folder, which contains the main source codes, there are two files, queueInterfaces.ts and queue.ts.

```
export type PriorityQueueComparator <T> = (a: T, b: T) => boolean;

export interface IPriorityQueueOptions <T>
{ comparator: PriorityQueueComparator <T>;
  initialElements?: T []; }

export interface IPriorityQueue <T> { size
  (): number; isEmpty (): boolean; peek ():
  T;
```

```

push (value: T): void;
pushMany (values: T []): void;
pop (): T; clear (): void; has
(value: T): boolean; values ():
T []; }

```

Table 13 - queueInterfaces.ts

In this file there is a type PriorityQueueComparator and interfaces IPriorityQueueOptions and IPriorityQueue. All are generic and exported.

PriorityQueueComparator shows what shape we want the compare function to be, which is necessary to initialize the priority queue.

IPriorityQueueOptions represents the object that the priority queue constructor receives. In addition to the comparator, it is possible to optionally send elements that will be in the priority order immediately after initialization.

IPriorityQueue presents the look of our priority order, shows us which public functions the priority order contains, which arguments receive those functions, and what they return.

4.8 Implementing Functions

The priority row in the src / queue.ts file is then implemented. Functions are not shown in the order in which they are implemented, nor how they are in the file, but so that each function that calls some other functions is explained only after the explanation of the functions that are called within it.

```

import
  {IPriorityQueue,
   PriorityQueueComparator,
   IPriorityQueueOptions,}
from './queueInterfaces';

export default class PriorityQueue <T> implements IPriorityQueue <T>
  {private heap: T []; private comparator: PriorityQueueComparator <T>;

  ...

```

Table 14 - src / queue.ts (part 1)

The priority order implements the interface intended for it. In addition to the fields from the interface, private fields have been added to it. So there is a generic array, which is used to implement the heap, and a comparator.

```
...
```

```

private getParent (index: number) {return
  (index + 1) >>> 1) - 1; }

private getLeftChild (index: number) {return
  (index << 1) + 1; }

private getRightChild (index: number) {return
  (index + 1) << 1; }

private compareByIndex (i: number, j: number) {return
  this.comparator (this.heap [i], this.heap [j]); }

private swap (i: number, j: number)
  {[this.heap [i], this.heap [j]] = [this.heap [j], this.heap [i]]; }

...

```

Table 15 - src / queue.ts (part 2)

Some auxiliary functions are needed. Thus, functions were created that will calculate the indices of his parent, as well as his left and right son. In order to perform these functions faster, binary operations were used.

In addition, a function has been created that returns the one that is more priority for the two indexes, as well as a function that replaces the values of the two nodes.

```

...
private siftUp ()
{let node = this.size () -
1; while (node > 0 && this.compareByIndex (node, this.getParent (node)))
{const parentNode = this.getParent (node); this.swap (node, parentNode);
node = parentNode; }}

/ **
*
* @param value element to be added to heap, adds it in O (log n) operations, n is size of heap *
@returns size of heap * / push (value: T) {this.heap.push (value); this.siftUp (); return this.size
(); }

...

```

Table 16 - src / queue.ts (part 3)

The paper implements a function for adding to the priority order, the push function. After adding to the stack it calls the siftUp (straighten up) function and returns the size of the priority row. This addition is performed in logarithmic time.

siftUp is a function that starts from a new element, located in a tree leaf, and climbs up as long as its priority is higher than its parent's priority, or until that element has climbed to the top of the tree.

```

...
/ **
*
* @param values elements to be added to heap, adds it in O (k * log n) operations, n is size of heap,
k is number of elements added
* @returns size of heap * /

pushMany (values: T [])
{values.forEach (value) =>
{this.push (value);}); return
this.size (); }

...

```

Table 17 - src / queue.ts (part 4)

There is also a `pushMany` function, which allows you to add a series of elements at once. It calls the `push` function for each element before returning the new priority order size. Therefore, the complexity of this function is $O(k * \log n)$, where n is the size of the priority row, even the size of the passed string, which should be entered in the priority row.

```
...
private siftDown (node = 0)
  (flight leftChild = this.getLeftChild (node);
  let rightChild = this.getRightChild (node);
  while (leftChild <this.size () &&
    this.compareByIndex (leftChild, node)) || (rightChild <this.size () &&
    this.compareByIndex (rightChild, node))) {const maxChild = rightChild
    <this.size () && this.compareByIndex (rightChild, leftChild)? rightChild:
    leftChild;

    this.swap (node, maxChild);

    node = maxChild;
    leftChild = this.getLeftChild (node);
    rightChild = this.getRightChild (node); }}
...
```

Table 18 - src / queue.ts (part 5)

Another private feature that is necessary is `siftDown`. Her task is to start from a knot, and as long as his priority is lower than that of one of his sons, to replace him with a more priority son. If we do not submit which node we want to start from, it will be assumed that we want to start from the head of the tree.

This function does its job in logarithmic time and is necessary when removing elements from the priority order, but also when creating a priority order with initial elements.

```
...
/ **
 *
 * @returns top of priority queue and removes it from priority queue in O (log n), if priority queue is
empty returns undefined
 * / pop
() {if (this.isEmpty ()) {
```

```
return undefined; }
```

```
const returnValue = this.peak ();
const lastIndexOfHeapArray = this.size () - 1; if
(lastIndexOfHeapArray > 0) {this.swap (0,
  lastIndexOfHeapArray); } this.heap.pop ();
this.siftDown (); return returnValue; }
```

```
...
```

Table 19 - src / queue.ts (part 6)

The next of the basic functions that had to be implemented was the pop function, the purpose of which was to remove elements from the priority order.

This function in logarithmic time, restores the highest priority element from the priority order, but also removes it, and reconstructs the order. If the row is empty this function returns undefined. Undefined in TypeScript is usually assigned to variables that have been declared but have not yet been assigned a value.

```
...
```

```
private buildHeap (array: T [])
{this.heap = JSON.parse (JSON.stringify (array)) as T []; for
(let i = Math.floor (array.length / 2) - 1; i >= 0; i--) {this.siftDown
  (i); }}
```

```
...
```

Table 20 - src / queue.ts (part 7)

The private buildHeap method will be used when building a heap, in case it is initialized with some initial elements. It for each element that is not a sheet calls the siftDown function. This is done in linear time.

```
...
```

```
/ **
 *
```

```
* @param options *
```

```
options.comparator: function used to compare elements; *
```

```
options.initialElements: elements to be put in priority queue initially in O (n) time * / constructor
```

```
(options: IPriorityQueueOptions (<T>)) {this.heap = []; this.comparator = options.comparator;
```

```
if (options.initialElements) this.buildHeap (options.initialElements); }
```

```
...
```

Table 21 - src / queue.ts (Part 8)

A constructor is also needed. It receives the required comparator and option through the initial elements and creates a priority row. In case the initial elements are sent, this function works in linear, and otherwise in constant time.

```
...
/ **
 *
 * @returns size of priority queue in O (1)
 * / size () {return this.heap.length; }

/ **
 *
 * @returns is priority queue empty in O (1)
 * / isEmpty () {return this.size () === 0; }
...
```

Table 22 - src / queue.ts (part 9)

Functions that return the size of the string and whether the string is empty are also implemented. Therefore, the size and isEmpty functions were created, both with constant complexity.

```
...
/ **
 *
 * @returns top of priority queue in O (1), if priority queue is empty returns undefined * /
 * peek (): T {return this.heap [0]? JSON.parse (JSON.stringify (this.heap [0])): undefined; }

/ **
 * clears priority queue in O (1)
 * /
clear (): void
{this.heap =
[]; }
```

...

Table 23 - src / queue.ts (part 10)

Functions that return the value of the highest priority element from the priority order are also needed, as well as a function that will empty the string.

The peek function will return the value of the highest priority element, at a constant time, by copying the value of the highest priority element. Copying is done in order to avoid that if it is an object, by later manipulation with the result of this function, the correctness of the structure is endangered.

The clear function will replace the heap with an empty string, leaving the deletion of elements from memory to the garbage collector, a process that removes all unused objects from memory in the background.

```
...
/ **
 * checks if value exists in priority queue in O (n) * /

has (value: T)
  (return !! this.heap.find (ele) => ele === value); }

/ **
 *
 * @returns all values of priority queue in O (n) * /
values () {return JSON.parse (JSON.stringify
(this.heap)) as T []; }

...
```

Table 24 - src / queue.ts (part 11)

Finally, the has and values functions are implemented.

has has a function to check if an element exists in priority order. This check is performed in linear time and if the elements of the row are reference types, this check is performed by reference.

values returns the values of all elements in the array by copying the value of each, much like a peek function. This function does not return elements in sorted order, so it works in linear complexity.

If you would like this function to return elements in sorted order, a pop function can be called until the priority row becomes empty. This process would be identical to the work of heap sort algorithms.

4.9 Building a Library

```
...
"main": "lib / queue.js",
"types": "lib / queue.d.ts",
"scripts": {
  ...
  "build": "tsc --project tsconfig.build.json",
  ...
},
...
```

Table 25 - changes to package.json

A script for building a project has been added to package.json. This script will let the tsc (TypeScript compiler) translate the files into JavaScript files.

Also, the package.json file indicates where the main file is, which contains functions that library users will be able to use, and where TypeScript types are located. These values are placed under the main and types fields.

```
npm run build
```

Table 26 - run build command

Now it is enough to run the build script and get the lib folder and the files in it: queue.d.ts, queue.js, queueInterfaces.d.ts and queueInterfaces.js.

.d.ts files are TypeScript declaration files. They contain lists of available types.

```
"use strict";
Object.defineProperty(exports, "__esModule", {value: true}); class
PriorityQueue {/ **
  *
  * @param options *
  options.comparator: function used to compare elements; *
  options.initialElements: elements to be put in priority queue initially in O (n) time * /
  constructor (options) {this.heap = []; this.comparator = options.comparator; if
  (options.initialElements) this.buildHeap (options.initialElements);

  } / **
  *
  * @returns size of priority queue in O (1) * /
```

```

size ()
  {return this.heap.length; } /
**

*

* @returns is priority queue empty in O (1)
* / isEmpty () {return this.size () === 0; } / **

*

* @returns top of priority queue in O (1), if priority queue is empty returns undefined * /
peek () {return this.heap [0]? JSON.parse (JSON.stringify (this.heap [0])): undefined; } / **
* clears priority queue in O (1) * / clear () {this.heap = []; } / ** * checks if value exists in
  priority queue in O (n) * / has (value) {return !! this.heap.find (ele) => ele === value); } /
**

*

* @returns all values of priority queue in O (n) * /
values () {return JSON.parse (JSON.stringify
(this.heap)); } buildHeap (array) (this.heap =
  JSON.parse (JSON.stringify (array)); for (let i =
  Math.floor (array.length / 2) - 1; i >= 0; i--)
  {this.siftDown (i); }

} / **
*

* @param value element to be added to heap, adds it in O (log n) operations, n is size of heap *
@returns size of heap

```

```

* /

push (value)
{this.heap.push (value);
this.siftUp (); return
this.size (); } / **

*

* @param values elements to be added to heap, adds it in O (k * log n) operations, n is size of heap,
k is number of elements added *
@returns size of heap * /

pushMany (values)
(values.forEach (value) =>
{this.push (value);}); return
this.size (); } / **

*

* @returns top of priority queue and removes it from priority queue in O (log n), if priority queue is
empty returns undefined

* / pop
() {if (this.isEmpty ())
{return undefined;

} const returnValue = this.peek ();
const lastIndexOfHeapArray = this.size () - 1; if
(lastIndexOfHeapArray > 0) {
this.swap (0, lastIndexOfHeapArray); }
this.heap.pop (); this.siftDown (); return
returnValue;

} getParent (index)
{return (index + 1) >>> 1) - 1; }
getLeftChild (index) {return (index <<
1) + 1; } getRightChild (index) {

return (index + 1) << 1; }
compareByIndex (i, j) {

```

```

    return this.comparator (this.heap [i], this.heap [j]); }
    swap (i, j) {[this.heap [i], this.heap [j]] = [this.heap [j],
    this.heap [i]];

    } siftUp ()
    {let node = this.size () -
    1; while (node > 0 && this.compareByIndex (node, this.getParent (node))) {
        const parentNode = this.getParent (node);
        this.swap (node, parentNode); node =
        parentNode; }

    } siftDown (node = 0)
    {let leftChild = this.getLeftChild (node);
    let rightChild = this.getRightChild (node);
    while (leftChild <this.size () && this.compareByIndex (leftChild, node)) ||
        (rightChild <this.size () && this.compareByIndex (rightChild, node))) {const
        maxChild = rightChild <this.size () && this.compareByIndex (rightChild, leftChild)?
rightChild: leftChild;
        this.swap (node, maxChild);
        node = maxChild; leftChild =
        this.getLeftChild (node); rightChild =
        this.getRightChild (node); }

    }

    } exports.default = PriorityQueue;

```

Table 27 - lib \ queue.js

In queue.js there is an implementation of functions, translated into

```

JavaScript. import {IPriorityQueue, IPriorityQueueOptions} from './
queueInterfaces'; export default class PriorityQueue <T> implements IPriorityQueue <T> {
    private heap;
    private comparator; /
    **
    *
    * @param options
    * options.comparator: function used to compare elements; *
    options.initialElements: elements to be put in priority queue initially in O (n) time * /
    constructor (options: IPriorityQueueOptions <T>); / **

    *
    * @returns size of priority queue in O (1)
    * /

```

```
size (): number; /
```

```
**
```

```
*
```

```
* @returns is priority queue empty in O (1) * /
```

```
isEmpty (): boolean; / **
```

```
*
```

```
* @returns top of priority queue in O (1), if priority queue is empty returns undefined * /
```

```
peek (): T; / ** * clears priority queue in O (1) * / clear (): void; / ** * checks if value exists in  
priority queue in O (n) * /
```

```
has (value: T): boolean; /
```

```
**
```

```
*
```

```
* @returns all values of priority queue in O (n) * /
```

```
values (): T []; private buildHeap; / **
```

```
*
```

```
* @param value element to be added to heap, adds it in O (log n) operations, n is size of heap *
```

```
@returns size of heap * / push (value: T): number; / **
```

```
*
```

```
* @param values elements to be added to heap, adds it in O (k * log n) operations, n is size of heap,  
k is number of elements added
```

```
* @returns size of heap * /
```

```
pushMany (values: T []): number; /
```

```
**
```

```
*
```

```
* @returns top of priority queue and removes it from priority queue in O (log n), if priority queue is  
empty returns undefined
```

```
* / pop
```

```
(): T; private
```

```
getParent; private getLeftChild;
```

```
private getRightChild;
private compareByIndex;
private swap; private siftUp;
private siftDown; }
```

Table 28 - lib \ queue.d.ts

The queue.d.ts file is a declaration file and contains a priority row type, which will be used by TypeScript users of this library.

```
export declare type PriorityQueueComparator <T> = (a: T, b: T) => boolean; export
interface IPriorityQueueOptions <T> {comparator: PriorityQueueComparator <T>;
  initialElements?: T []};

} export interface IPriorityQueue <T> {size
  (): number; isEmpty (): boolean; peek
  (): T; push (value: T): void; pushMany
  (values: T []): void; pop (): T; clear ():
  void; has (value: T): boolean; values ():
  T []; }
```

Table 29 - lib \ queueInterfaces.d.ts

The queueInterfaces.d.ts file contains the types and interfaces formed for TypeScript from the queueInterfaces.ts file.

```
"use strict";
Object.defineProperty (exports, "__esModule", {value: true});
```

Table 30 - lib \ queueInterfaces.js

queueInterfaces.js is practically empty and contains only the syntax for exporting between different modules.

That we want to enable use in different types of modules has already been indicated by the option "esModuleInterop" in the tsconfig file.

4.10 Adding a LICENSE file Computer software is

often accompanied by licenses, which tell you what can be done with that software. Open source code is accompanied by licenses that allow free use, modification and distribution of code. The most common open source licenses are ISC, MIT, BSD, but there are others. In this project, ISC was chosen, which is essentially similar to the others, but written in simpler language.

When creating a library with npm, a choice of licensing was offered. In this step, only the license name would be added to the package.json file, but you would not get the generated LICENSE file. If no license is selected then, it can be easily added to package.json later.

```
...  
"license": "ISC",  
...
```

Table 31 - license in package.json

Then it is necessary to create a LICENSE file. You can use the open-source library from npm generate-license (<https://www.npmjs.com/package/generate-license>), version 1.0.0.

```
npm install --global generate generate-license
```

Table 32 - Command to install the licensing package

It is first necessary to install the generate and generate-license packages globally. The generate package is a project by the same authors that offers a tool used via the line command, and uses generators (such as generate license) to generate certain types of files. In addition to the license generator, there are various generators for README files, test files, files for integration with Travis, but also various other types of files.

```
gen license: isc
```

Table 33 - Command to generate a license file

The command from Table 33 created a LICENSE file for the project, with the ISC declaration.

By accessing this file via GitHub, it can be seen in simple terms that the author allows the commercial use, distribution, modification and private use of this code. It is also noted that the license contains restrictions that do not provide any guarantees, including a guarantee of reliability.

4.11 Creating examples

To make it easier for library users to use, examples of how to use it have been added. Examples have been made for use in both JavaScript and TypeScript.

4.11.1 Creating examples in JavaScript

In the root directory of the library, in the sample-projects folder, which was created to house test projects, a folder named project example was created. In this case, the folder is called "javascript".

```
npm init -y
```

Table 34 - JavaScript project initialization command

The command from Table 34 started the creation of the application. The "-y" flag tells npm to accept all default settings. Then the created package.json for the project example is obtained.

```
{
  "name": "javascript",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\ Error: no test specified \\&& exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Table 35 - package.json is an example of a project

```
npm and priority-queue-with-custom-comparator
```

Table 36 - command to install a previously created package

```
{
  "name": "test-library-with-js",
  "version": "1.0.0",
  "description": "",
  "type": "module",
  "main": "index.js",
  "scripts": {
    "start": "node ./index.js",
  }
}
```



```

    "test": "echo \" Error: no test specified \" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "priority-queue-with-custom-comparator": "^ 1.0.2"
  }
}

```

Table 37 - package.json is an example of a project after installing a package

The created library is installed and it is automatically added to the package.json file. In addition, certain modifications have been made to the package.json file, such as changing the project name and adding a script to start the project.

```

import PriorityQueue from 'priority-queue-with-custom-comparator';

const q = new PriorityQueue.default ({
  comparator: (a, b) => {
    return a - b < 0;
  },
});

q.pushMany ([- 1, 4, 8, -9]);
q.push (2);

console.log ('Queue size:', q.size ());

```

Table 38 - index.js

Then an index.js file was added which contains the basic work with the library. This is in this case creating a priority row, adding to it, and printing its size.

In order to use this library in JavaScript with ECMAScript modules, the `PriorityQueue.default` syntax must be used when importing, because the created library was made for CommonJS modules, which is the Node.js standard.

```
FROM node: 14-alpine

WORKDIR /usr/src/app

COPY ["package.json", "package-lock.json", "."]

RUN npm install --silent && mv node_modules ../

COPY . .

RUN chown -R node /usr/src/app

USER node

CMD ["node", "index.js"]
```

Table 39 - Dockerfile is an example of a project

```
** /.dockerignore
** /.vs
** /.vscode
** /docker-compose *
** /compose *
** /Dockerfile *

** /node_modules

README.md
```

Table 40 - .dockerignore is an example of a project

To make it easier for users of the created library to start an example project, a Dockerfile has been added from which a Docker image can be created. The `.dockerignore` file contains files that do not need to be used when creating an image. These files can be created manually, but the easier option is to use the Docker extension if VS Code is used.

So when creating an image, Docker will download the compressed image Node.js, version 14, and adjust the working folder within the image. It then copies `package.json` and `package-lock.json` and installs all the dependencies, and moves the `node_modules` folder (Docker's recommendation to avoid various OS related problems). Other files are copied, the owner of all files is changed, transferred to another user and the project is started.

4.11.2 Creating examples in TypeScript

NestJS, one of the most popular Node.js frameworks, which provides full support for TypeScript, was used to create an example project in TypeScript.

In the root directory of the library, in the sample-projects folder, which was created to house test projects, a folder named project example was created. In this case, the folder is called "nestjs typescript".

```
npm and -g @ nestjs / cli
```

Table 41 - Nest installation command

In order to use this framework, it is necessary to install it globally on the development machine.

```
nest new nestjs-typescript
```

Table 42 - command to create a new nest project

Then an example project was created through NestJS. When creating, a choice of package manager to be used was offered. The created library supports work with all package managers, but in this work it was chosen to work with npm.

NestJS creates configuration files for TypeScript, eslint, nest, creates git files, README file, package.json and package-lock.json, src folder containing codes, test folder for tests but also node_modules. In the src folder, it creates a main.ts file from which the project is started, as well as one module, an app module that runs from the main file. For the app module, it creates a module file, a controller file, a service file, and a test file.

In order to avoid conflicts with the library version control system, the created .git folder and the .gitignore file were removed. .Prettierrc and .eslintrc.js have been removed so that you do not have problems with linking the library.

The tight file for app modules has also been removed, as it is not used in this example. The example is placed directly in the main.ts file.

```
npm and priority-queue-with-custom-comparator
```

Table 43 - command to install a previously created package

Then the created library is installed.

```
/* import {NestFactory} from '@stjs / core';

import {AppModule} from './app.module';*/

import PriorityQueue from 'priority-queue-with-custom-comparator';

async function bootstrap () {
```

```

/ * const app = await NestFactory.create (AppModule);

await app.listen (3000); * /

const q = new PriorityQueue <number> ({
  comparator: (a, b) => {
    return a - b <0;
  },
});

q.pushMany ([- 1, 4, 8, -9]);

q.push (2);

console.log ('Queue size:', q.size ());
}

bootstrap ();

```

Table 44 - main.ts

The main.ts file contains the basic work with the library. This is in this case creating a priority row, adding to it, and printing its size. The syntax for creating and running app modules has been commented out. This is not a proper use of the NestJS framework but is a good enough example to show how to work with this library in TypeScript.

```

FROM node: 14-alpine

WORKDIR / usr / src / app

COPY ["package.json", "package-lock.json *", ". /"]

RUN npm i -g --silent @ nestjs / cli @ 8.1 && npm install --silent && mv node_modules ../

COPY . .

RUN chown -R node / usr / src / app

USER node

CMD ["npm", "start"]

```

Table 45 - Dockerfile ts example of a project

```

** /. dockerignore

** /. vs

```

```

** /. vscode

** / docker-compose *

** / compose *

** / Dockerfile *

** / node_modules

README.md

```

Table 46 - .dockerignore ts example of a project

Finally, files for Docker are added. The docker file for this project will be similar to the one for javascript, the only difference will be that NestJS will also be installed.

4.12 Test Writing and Test Coverage

Writing tests is a popular practice in software development that is gaining momentum. This makes the code more reliable, readable and easier to change. The most popular types of tests are unit, integration, end-to-end, smoke, but there are many others. Developers usually write unit, integration and end-to-end tests, while other teams write many types of tests.

Unit tests have been implemented within this library, because it only makes sense to implement them in such a project. Unit tests are tests that test small units of code, most often testing whether the right input functions return the right output and whether they cause real side effects.

This project has one hundred percent coverage of unit tests of tested pieces of code, ie queue.ts file. Although as much test coverage as possible is desirable, it is not a guarantee that there is no error in the tests, because the tests can still skip an important case, or expect the wrong result of an operation.

There are several testing frameworks for JavaScript, and Jest was used in this project.

```

{
  "transform": {
    "^. +\\. (t | j) sx? $": "ts-jest"
  },
  "testRegex": "(/__tests___/.*\\.)(spec))\\. (js|ts)$",
  "moduleFileExtensions": [
    "ts",

```

```

    "tsx",

    "JS",

    "jsx",

    "json",

    "node"

  ],

  "collectCoverageFrom": [

    "src / queue.ts",

    "! ** / __ tests __ / **",

    "! ** / node_modules / **"

  ]

}

```

Table 47 - jestconfig.json

The rules that will use Yes are defined in jestconfig.json. Jest was told which files to use for testing, where the tests were, and which files to measure test coverage.

Since the tests are more or less similar, only a part of the tests in the paper is shown, while the rest can be found in the git repository listed at the beginning of the paper.

```

export const defaultMaxComparator = (a: number, b: number): boolean => {

  return a > b;

};

```

Table 48 - __tests__ / test.helper.ts

An auxiliary file for tests has been created, from which the compare function for integers is exported, which sorts them from smaller to larger.

```

import PriorityQueue from '../src/queue';

import {defaultMaxComparator} from './test.helper';

test('initial state (created with initialElements)', () => {

  const numberPriorityQueue = new PriorityQueue <number> ({

```

```

    comparator: defaultMaxComparator,
    initialElements: [2, 3, 1],
  });

  expect (numberPriorityQueue.size ()). toBe (3);

  numberPriorityQueue.clear ();

  expect (numberPriorityQueue.size ()). toBe (0);
});

test ('initial state (created without initialElements)', () => {
  const numberPriorityQueue = new PriorityQueue <number> ({
    comparator: defaultMaxComparator,
  });

  expect (numberPriorityQueue.size ()). toBe (0);

  numberPriorityQueue.clear ();
  vr
  expect (numberPriorityQueue.size ()). toBe (0);
});

```

Table 49 - __tests__ / clear.spec.ts

Tests for clear function are extremely simple. The clear function over the created priority order, created with and without initial elements, is tested. In these tests, the size function is "expected" to return zero after the clear function is called.

```

import PriorityQueue from '../src/queue';
import {defaultMaxComparator} from './test.helper';

```

```

afterEach () => {
  jest.clearAllMocks ();
};

test ('initial state (created with initialElements)', () => {
  const numberPriorityQueue = new PriorityQueue <number> ({
    comparator: defaultMaxComparator,
    initialElements: [2, 6, 7],
  });
  expect (numberPriorityQueue.values (). toString ()). toBe ([7, 6, 2].toString ());
});

test ('initial state (created without initialElements)', () => {
  const numberPriorityQueue = new PriorityQueue <number> ({
    comparator: defaultMaxComparator,
  });
  expect (numberPriorityQueue.values (). toString ()). toBe ([] . toString ());
});

```

Table 50 - __tests__ / constructor.spec.ts

To test the constructor, simply call the values function after creating the priority row, which should return a string containing the same elements as the one passed for the initial values. If it was not, then we expect the values to return an empty string.

```

import PriorityQueue from '../src/queue';
import {defaultMaxComparator} from './test.helper';

test ('initial state (created without initialElements) and some added', () => {
  const numberPriorityQueue = new PriorityQueue <number> ({

```



```
    comparator: defaultMaxComparator,  
  });  
  
  expect (numberPriorityQueue.has (5)). toBe (false);  
  expect (numberPriorityQueue.has (2)). toBe (false);  
  expect (numberPriorityQueue.has (0)). toBe (false);  
  expect (numberPriorityQueue.has (-1)). toBe (false);  
  expect (numberPriorityQueue.has (6)). toBe (false);  
  expect (numberPriorityQueue.has (-2)). toBe (false);  
  expect (numberPriorityQueue.has (-9)). toBe (false);  
  expect (numberPriorityQueue.has (58)). toBe (false);  
  expect (numberPriorityQueue.values (). toString ()). toBe ([] . toString ());  
  expect (numberPriorityQueue.size ()). toBe (0);  
  
  numberPriorityQueue.push (5);  
  
  expect (numberPriorityQueue.has (5)). toBe (true);  
  expect (numberPriorityQueue.has (2)). toBe (false);  
  expect (numberPriorityQueue.has (0)). toBe (false);  
  expect (numberPriorityQueue.has (-1)). toBe (false);  
  expect (numberPriorityQueue.has (6)). toBe (false);  
  expect (numberPriorityQueue.has (-2)). toBe (false);  
  expect (numberPriorityQueue.has (-9)). toBe (false);  
  expect (numberPriorityQueue.has (58)). toBe (false);  
  expect (numberPriorityQueue.has (185)). toBe (false);  
  expect (numberPriorityQueue.has (201)). toBe (false);  
  expect (numberPriorityQueue.values (). toString ()). toBe ([5] . toString ());  
  expect (numberPriorityQueue.size ()). toBe (1);
```

```
numberPriorityQueue.push (-1);
```

```
expect (numberPriorityQueue.has (5)). toBe (true);  
expect (numberPriorityQueue.has (2)). toBe (false);  
expect (numberPriorityQueue.has (0)). toBe (false);  
expect (numberPriorityQueue.has (-1)). toBe (true);  
expect (numberPriorityQueue.has (6)). toBe (false);  
expect (numberPriorityQueue.has (-2)). toBe (false);  
expect (numberPriorityQueue.has (-9)). toBe (false);  
expect (numberPriorityQueue.has (58)). toBe (false);  
expect (numberPriorityQueue.has (185)). toBe (false);  
expect (numberPriorityQueue.has (201)). toBe (false);  
expect (numberPriorityQueue.values (). toString ()). toBe ([5, -1].toString ());  
expect (numberPriorityQueue.size ()). toBe (2);
```

```
numberPriorityQueue.push (2);
```

```
expect (numberPriorityQueue.has (5)). toBe (true);  
expect (numberPriorityQueue.has (2)). toBe (true);  
expect (numberPriorityQueue.has (0)). toBe (false);  
expect (numberPriorityQueue.has (-1)). toBe (true);  
expect (numberPriorityQueue.has (6)). toBe (false);  
expect (numberPriorityQueue.has (-2)). toBe (false);  
expect (numberPriorityQueue.has (-9)). toBe (false);  
expect (numberPriorityQueue.has (58)). toBe (false);  
expect (numberPriorityQueue.has (185)). toBe (false);  
expect (numberPriorityQueue.has (201)). toBe (false);
```

```
expect (numberPriorityQueue.values (). toString ()). toBe ([5, -1, 2] .toString ());  
expect (numberPriorityQueue.size ()). toBe (3);  
  
numberPriorityQueue.push (6);  
  
expect (numberPriorityQueue.has (5)). toBe (true);  
expect (numberPriorityQueue.has (2)). toBe (true);  
expect (numberPriorityQueue.has (0)). toBe (false);  
expect (numberPriorityQueue.has (-1)). toBe (true);  
expect (numberPriorityQueue.has (6)). toBe (true);  
expect (numberPriorityQueue.has (-2)). toBe (false);  
expect (numberPriorityQueue.has (-9)). toBe (false);  
expect (numberPriorityQueue.has (58)). toBe (false);  
expect (numberPriorityQueue.has (185)). toBe (false);  
expect (numberPriorityQueue.has (201)). toBe (false);  
expect (numberPriorityQueue.values (). toString ()). toBe  
  [6, 5, 2, -1] .toString (),  
);  
expect (numberPriorityQueue.size ()). toBe (4);  
  
numberPriorityQueue.push (6);  
  
expect (numberPriorityQueue.has (5)). toBe (true);  
expect (numberPriorityQueue.has (2)). toBe (true);  
expect (numberPriorityQueue.has (0)). toBe (false);  
expect (numberPriorityQueue.has (-1)). toBe (true);  
expect (numberPriorityQueue.has (6)). toBe (true);  
expect (numberPriorityQueue.has (-2)). toBe (false);
```

```
expect (numberPriorityQueue.has (-9)). toBe (false);
expect (numberPriorityQueue.has (58)). toBe (false);
expect (numberPriorityQueue.has (185)). toBe (false);
expect (numberPriorityQueue.has (201)). toBe (false);
expect (numberPriorityQueue.values (). toString ()). toBe
  [6, 6, 2, -1, 5] .toString (),
);
expect (numberPriorityQueue.size ()). toBe (5);

numberPriorityQueue.push (-2);

expect (numberPriorityQueue.has (5)). toBe (true);
expect (numberPriorityQueue.has (2)). toBe (true);
expect (numberPriorityQueue.has (0)). toBe (false);
expect (numberPriorityQueue.has (-1)). toBe (true);
expect (numberPriorityQueue.has (6)). toBe (true);
expect (numberPriorityQueue.has (-2)). toBe (true);
expect (numberPriorityQueue.has (-9)). toBe (false);
expect (numberPriorityQueue.has (58)). toBe (false);
expect (numberPriorityQueue.has (185)). toBe (false);
expect (numberPriorityQueue.has (201)). toBe (false);
expect (numberPriorityQueue.values (). toString ()). toBe
  [6, 6, 2, -1, 5, -2] .toString (),
);
expect (numberPriorityQueue.size ()). toBe (6);

numberPriorityQueue.push (-9);
```

```
expect (numberPriorityQueue.has (5)). toBe (true);
expect (numberPriorityQueue.has (2)). toBe (true);
expect (numberPriorityQueue.has (0)). toBe (false);
expect (numberPriorityQueue.has (-1)). toBe (true);
expect (numberPriorityQueue.has (6)). toBe (true);
expect (numberPriorityQueue.has (-2)). toBe (true);
expect (numberPriorityQueue.has (-9)). toBe (true);
expect (numberPriorityQueue.has (58)). toBe (false);
expect (numberPriorityQueue.has (185)). toBe (false);
expect (numberPriorityQueue.has (201)). toBe (false);
expect (numberPriorityQueue.values (). toString ()). toBe
  [6, 6, 2, -1, 5, -2, -9] .toString (),
);
expect (numberPriorityQueue.size ()). toBe (7);
```

```
numberPriorityQueue.push (58);
```

```
expect (numberPriorityQueue.has (5)). toBe (true);
expect (numberPriorityQueue.has (2)). toBe (true);
expect (numberPriorityQueue.has (0)). toBe (false);
expect (numberPriorityQueue.has (-1)). toBe (true);
expect (numberPriorityQueue.has (6)). toBe (true);
expect (numberPriorityQueue.has (-2)). toBe (true);
expect (numberPriorityQueue.has (-9)). toBe (true);
expect (numberPriorityQueue.has (58)). toBe (true);
expect (numberPriorityQueue.has (185)). toBe (false);
expect (numberPriorityQueue.has (201)). toBe (false);
expect (numberPriorityQueue.values (). toString ()). toBe
```

```
[58, 6, 2, 6, 5, -2, -9, -1] .toString (),  
);  
expect (numberPriorityQueue.size ()). toBe (8);  
  
numberPriorityQueue.push (185);  
  
expect (numberPriorityQueue.has (5)). toBe (true);  
expect (numberPriorityQueue.has (2)). toBe (true);  
expect (numberPriorityQueue.has (0)). toBe (false);  
expect (numberPriorityQueue.has (-1)). toBe (true);  
expect (numberPriorityQueue.has (6)). toBe (true);  
expect (numberPriorityQueue.has (-2)). toBe (true);  
expect (numberPriorityQueue.has (-9)). toBe (true);  
expect (numberPriorityQueue.has (58)). toBe (true);  
expect (numberPriorityQueue.has (185)). toBe (true);  
expect (numberPriorityQueue.has (201)). toBe (false);  
expect (numberPriorityQueue.values (). toString ()). toBe  
[185, 58, 2, 6, 5, -2, -9, -1, 6] .toString (),  
);  
expect (numberPriorityQueue.size ()). toBe (9);  
  
numberPriorityQueue.push (201);  
  
expect (numberPriorityQueue.has (5)). toBe (true);  
expect (numberPriorityQueue.has (2)). toBe (true);  
expect (numberPriorityQueue.has (0)). toBe (false);  
expect (numberPriorityQueue.has (-1)). toBe (true);  
expect (numberPriorityQueue.has (6)). toBe (true);
```

```

expect (numberPriorityQueue.has (-2)). toBe (true);
expect (numberPriorityQueue.has (-9)). toBe (true);
expect (numberPriorityQueue.has (58)). toBe (true);
expect (numberPriorityQueue.has (185)). toBe (true);
expect (numberPriorityQueue.has (201)). toBe (true);
expect (numberPriorityQueue.values (). toString ()). toBe
  [201, 185, 2, 6, 58, -2, -9, -1, 6, 5] .toString (),
);
expect (numberPriorityQueue.size ()). toBe (10);
});

```

Table 51 - __tests__ / push.spec.ts

The push function is tested by checking after each call of the push function whether the priority row contains the correct elements and whether the priority row is the right size.

```

...
"scripts": {
...
  "test": "is --config jestconfig.json",
  "test: coverage": "is --config jestconfig.json --coverage",
},
...

```

Table 52 - scripts related to tests in package.json

A script has been added to package.json to run tests, and create test coverage reports.

After running the test: coverage script, you get a coverage folder, which contains metadata about the coverage of the tests, but also an overview web page with displayed information about it.

4.13 Writing a README file

In order for others to know what this library is, how it is used, what its functionalities are, but also other information, a README.md file was created. It is a fairly popular type of file written in the Markdown language.

```
# priority-queue-with-custom-comparator
```

```
! [npm] (https://img.shields.io/npm/v/priority-queue-with-custom-comparator)
```

```
[! [npm] (https://img.shields.io/npm/dm/priority-queue-with-custom-comparator.svg)] (https://
```

```
www.npmjs.com/package/priority-queue-with-custom-comparator)
```

```
[! [npm] (https://img.shields.io/badge/node-%3E=%206.0-blue.svg)] (https://www.npmjs.com/package/
```

```
priority-queue-with-custom-comparator)
```

```
! [GitHub
```

```
repo
```

```
size] (https://img.shields.io/github/repo-size/dejtor/priority-queue-with-custom-comparator) (https://
```

```
! [npm]
```

```
img.shields.io/npm/l/priority-queue-with-custom-comparator) commit] (https://img.shields.io/github/
```

```
last
```

```
last-commit/dejtor/priority-queue-with-custom-comparator)
```

```
"
```

```
[Travis (.com)] (https://img.shields.io/travis/com/dejtor/priority-queue-with-custom-comparator)
```

```
"img      src = "https://user-images.githubusercontent.com/6517308/121813242-859a9700-cc6b-11eb
99c0-49e5bb63005b.jpg">
```

Priority queue implemented using Heap data structure, allows using custom comparator function.

```
# Contents
```

```
- [Example] (# Example)
```

```
- [TS] (# TS)
```

```
- [JS] (# JS)
```

```
- [Functions] (# Functions)
```

```
## Example:
```

```
### TS
```

```
import PriorityQueue from 'priority-queue-with-custom-comparator';
```



```

class Rand {
  num: number;
}

const numberPriorityQueue = new PriorityQueue <number> ({
  comparator: (a, b) => {
    return a - b < 0;
  },
  initialElements: [3, 1],
});

numberPriorityQueue.pushMany ([- 1, 4, 8, -9]);
numberPriorityQueue.push (2);

console.log ('top of queue', numberPriorityQueue.pop ());
console.log ('top of queue', numberPriorityQueue.peek ());
console.log (
  'size after inserting 1, 2 and 3',
  numberPriorityQueue.pushMany ([1, 2, 3]),
);

const classPriorityQueue = new PriorityQueue <Rand> ({
  comparator: (a, b) => a.num > b.num,
});

classPriorityQueue.pushMany ([
  {num: 5},
  {num: 1},

```

```

    {num: -9},
    {num: 11},
    {num: 15},
    {num: 51},
    {num: 155},
  ]);

  console.log ('classPriorityQueue:', classPriorityQueue.values ());

  const stringPriorityQueue = new PriorityQueue <string> ({
    comparator: (a, b) => a.length > b.length,
  });

  stringPriorityQueue.pushMany ('abcd', 'a', 'abcdeef', 'string');

  console.log ('stringPriorityQueue:', stringPriorityQueue.values ());
  ...

  ### JS

  ...

  import PriorityQueue from 'priority-queue-with-custom-comparator'

  class Rand {
    num;
  }

  const numberPriorityQueue = new PriorityQueue.default ({

```

```

    comparator: (a, b) => {
      return a - b < 0;
    },
    initialElements: [3, 1],
  });
numberPriorityQueue.pushMany([-1, 4, 8, -9]);
numberPriorityQueue.push(2);

console.log('top of queue', numberPriorityQueue.pop());
console.log('top of queue', numberPriorityQueue.peek());
console.log('size after inserting 1, 2 and 3', numberPriorityQueue.pushMany([1, 2, 3]));

const classPriorityQueue = new PriorityQueue.default({comparator: (a, b) => a.num > b.num});

classPriorityQueue.pushMany([
  {num: 5},
  {num: 1},
  {num: -9},
  {num: 11},
  {num: 15},
  {num: 51},
  {num: 155},
]);

console.log('classPriorityQueue:', classPriorityQueue.values());

const stringPriorityQueue = new PriorityQueue.default({comparator: (a, b) => a.length > b.length});

stringPriorityQueue.pushMany('abcd', 'a', 'abcdeef', 'string');

```

```
console.log ('stringPriorityQueue:', stringPriorityQueue.values ());
```

```
...  
## Functions:
```

```
...  
  
/ **
```

```
*  
  

```

```
* @param options
```

```
* options.comparator: function used to compare elements;
```

```
* options.initialElements: (optional) elements to be put in priority queue initially in O (n) time
```

```
* /
```

```
constructor (options: PriorityQueueOptions <T>);
```

```
/ **
```

```
*  
  

```

```
* @returns size of priority queue in O (1)
```

```
* /
```

```
size (): number;
```

```
/ **
```

```
*  
  

```

```
* @returns is priority queue empty in O (1)
```

```
* /
```

```
isEmpty (): boolean;
```

```
/ **
```

```
*  
  

```

```
* @returns top of priority queue in O (1), if priority queue is empty returns undefined
```

```
* /
```

```

peek (): T;
/ **
* clears priority queue in O (1)
* /

clear (): void;
/ **
* checks if value exists in priority queue in O (n)
* /

has (value: T): boolean;
/ **
*
* @returns all values of priority queue in O (n)
* /

values (): T [];
/ **
*
* @param value element to be added to heap, adds it in O (log n) operations, n is size of heap
* @returns size of heap
* /

push (value: T): number;
/ **
*
* @param values elements to be added to heap, adds it in O (k * log n) operations, n is size of heap,
k is number of elements added
* @returns size of heap
* /

pushMany (values: T []): number;
/ **

```

```

*

* @returns top of priority queue and removes it from priority queue in O (log n), if priority queue is
empty returns undefined

* /

pop (): T;

```

Table 53 - README.md

The README file contains the name of the library, shields, a label that supports both JavaScript and TypeScript, a brief description, content, examples in JavaScript and TypeScript, but also descriptions and complexities of all public or exported functions. Shields have been put up showing the latest version of the library, the number of monthly downloads, the required version of Node, the size of the repository, the license, the date of the last commitment, and whether all Travis tests pass.

4.14 Using Travis to run tests automatically

In order for tests to be performed after each push of the code on GitHub, and publicly shown that they pass successfully, Travis was used. This gives users extra security when using this library.

```

language: node_js

node_js:
- 14

```

Table 54 - .travis.yml

The Travis file is written in yaml format and it was enough to mention which technology and which version is used. We also need to have a test script in the package.json file, which we have already added.

Then it was necessary to register at <https://www.travis-ci.com/>, via an existing GitHub account. On the Travis site, activation is performed, and the choice of which GitHub repositories want to integrate with Travis, in this case a created library. This completes the integration, and after each subsequent push, Travis performs tests, notifies the author of the result, and maintains the shield used in the README file accordingly.

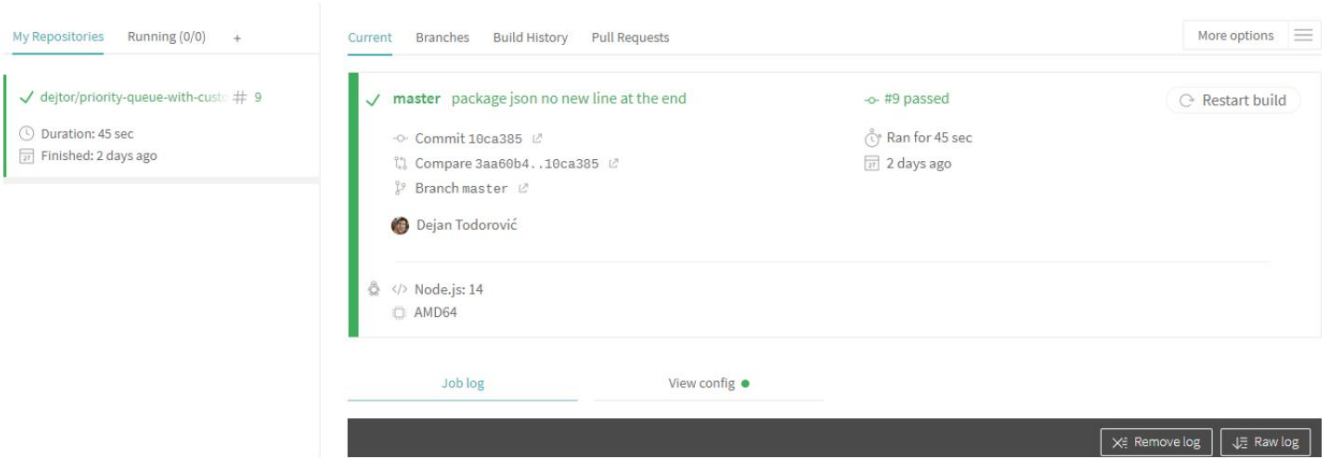


Figure 4 - Travis homepage

The results can be found on the Travis homepage, along with detailed logs, execution history but also other details and options.

4.15 Publishing the Library

In the end, it was necessary to publish the library on the npm register.

```
npm publish
```

Table 55 - command to publish the library

The publish command publishes packages on the npm register, after which the library can be found on the site of npmjs, yarn, but also other package managers for JavaScript packages.

If we want to improve and change the library, it is recommended to change its version.

```
npm version <major | minor | patch>
```

Table 56 - Library versioning command

Depending on the type of change, the major, minor or patch number of the library changes, ie the first, second or third number in the version. If a change is made, such that the code that worked with the previous version of the library could stop working by switching to a newer version, the major changes. If only some new functionality is added, the minor is changed, and if only some minor fixes or patches are made, the patch is changed.

```
...

"scripts": {
  ...
```

```

"prepare": "npm run build",
"prepublishOnly": "npm test && npm run lint",
"verification": "npm run lint",
"version": "npm run format && git add -A src",
"postversion": "git push && git push --tags"
},
...

```

Table 57 - Versioning scripts in package.json

Scripts have been added to run when publishing and versioning the library.

Thus, before the code is published, it will be tested whether it can be successfully built, whether the tests pass, and whether the linter throws some errors.

When versioning the code, the code will also be lined, formatted and pushed to a remote branch.

```

...

"repository": {
  "type": "git",
  "url": "https://github.com/dejtor/priority-queue-with-custom-comparator.git"
},
"files": [
  "lib / ** / *"
],
"keywords": [
  "priority queue",
  "heap",
  "custom-comparator",
  "comparator",
  "data structure",
  "data structures",
  "priority",

```


"queue"
,
...

Table 58 - add-ons in the package.json file

In order to better describe this library on GitHub and the pages associated with the npm registry, a description of where the remote repository is located has been added to the package.json file, as well as keywords for the library. It also added which files should be whitelisted in the npm release, to avoid adding unnecessary files for library users.

5 Conclusion

This paper shows the procedure for creating a library for Node.js and publishing it to the npm registry. Various good practices were used, test writing, example and README file writing, Continuous Integration tools, Docker files created, license files, linters and formatters used. The author considered all this as a necessary, but not sufficient, condition to make a good library for npm. Node is not the only technology that allows users to create and publish their own libraries, and the process for other technologies can have many similarities.

6 Bibliography

<https://medium.com/selleo/top-trends-in-node-js-to-watch-in-2021-d94ff38cc31e> (accessed 4. October 2021 19:00 CEST)

<https://docs.travis-ci.com/user/for-beginners/> (accessed 4 October 2021 19:15 CEST)

https://en.wikipedia.org/wiki/Travis_CI (accessed 4 October 2021 19:20 CEST)

[https://en.wikipedia.org/wiki/npm_\(software\)](https://en.wikipedia.org/wiki/npm_(software)) (accessed 4 October 2021 19:30 CEST)

<https://en.wikipedia.org/wiki/Node.js> (accessed 4 October 2021 19:40 CEST)

<https://en.wikipedia.org/en-us/JavaScript> (accessed 4 October 2021 19:50 CEST)

https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript (access time 4. October 2021 19:55 CEST)

<https://en.wikipedia.org/wiki/ESLint> (accessed 7 October 2021 22:40 CEST)

https://sr.wikipedia.org/sr-el/%D0%A0%D0%B5%D0%B4_%D1%81%D0%B0_%D0%BF%D1%80%D0%B8%D0%BE%D1%80%D0%B8%D1%82%D0%B5%D1%82%D0%BE%D0%BC

(accessed October 7, 2021 10:50 PM CEST)

[https://sr.wikipedia.org/wiki/Hip_\(struktura_podataka\)](https://sr.wikipedia.org/wiki/Hip_(struktura_podataka)) (accessed October 7, 2021, 10:55 PM CEST)