Lab Report

Operating Systems Laboratory: Assignment 6

Yatindra Indoria, Srishty Gandhi, Chirag Ghosh, Sarita Singh
20CS30060 20CS30052 20CS10020 20CS10053

Indian Institute of Technology Kharagpur

a. What is the structure of your internal page table? Why?

```
typedef struct seg {
    size_t size;
    void *addr;
} segment;
```

The page table contains the addresses and sizes of the memory blocks available, and is capable of **O(n)** search for a **first-fit** memory block, with memory size.

A salient feature is, the unequal size of blocks being supported. This allows an **O(1) free** operation on allocated blocks. The address is updated (the address is incremented or, the block is removed in case of deletion).

b. What are additional data structures/functions used in your library? Describe all with justifications.

```
typedef struct list {
    Node * root;
    size_t nelems;
} List;
```

We defined a struct called List which represents a doubly linked list data structure.
The List struct has two fields:
root: a pointer to the first (or root) Node in the linked list.
nelems: a size_t variable that stores the number of elements (or nodes) in the linked list.

The **nelems** field is useful for **O(1)** size estimates when required. This field was also used for the efficient logging of the memory footprint.

```
typedef struct node {
    struct node * lnode;
    struct node * rnode;
    int value;
} Node;
```

In this Node struct, the lnode and rnode pointers represent the previous and next nodes in the linked list, respectively. The value field stores the value of the current node.

```
typedef struct symtable {
    List *lst;
    unsigned int depth;
} sentry;
```

The **symbol-table** is designed keeping recursion in mind. The depth field is the depth of the call-stack, maintained as **global state** by the library. This is used for differentiation.

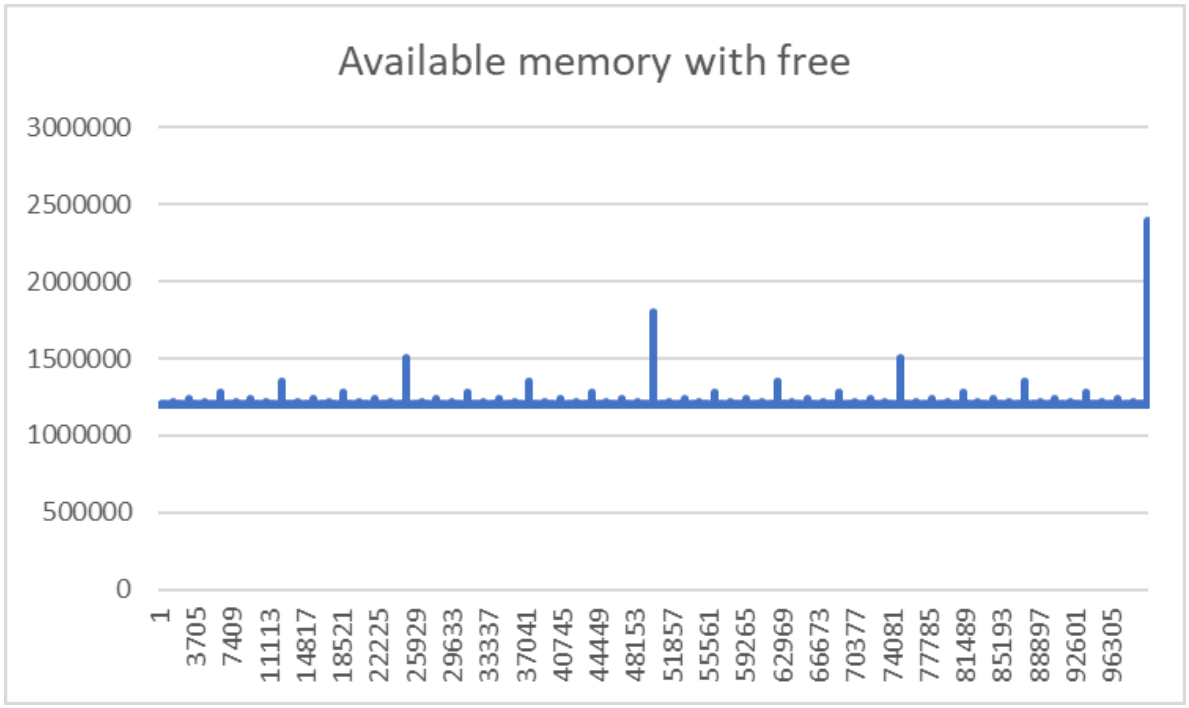
We also use the addresses of the **List** variables instead of names to use numerical lookup/equality check for efficiency.
In case of a use-case switch, we can **store -1 in depth** to indicated invalidity of a particular entry, and it can be garbage collected.
Note: To be noted the use of the above symbol table requires that we call a function **initFunc**, at function start, and that we cann the function **freeElem** with a **NULL** argument, to remove all the local variables at exit.

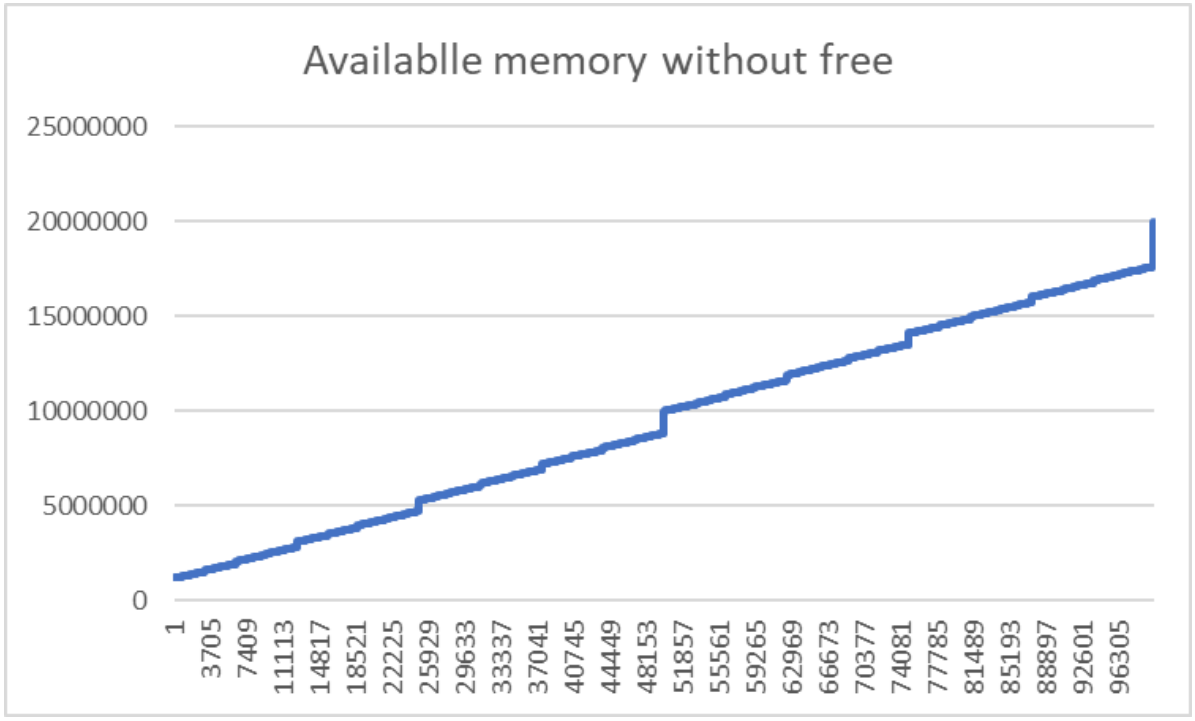
c. What is the impact of freeElem() for merge sort. Report the memory footprint and running time with and without freeElem (average over 100 runs).

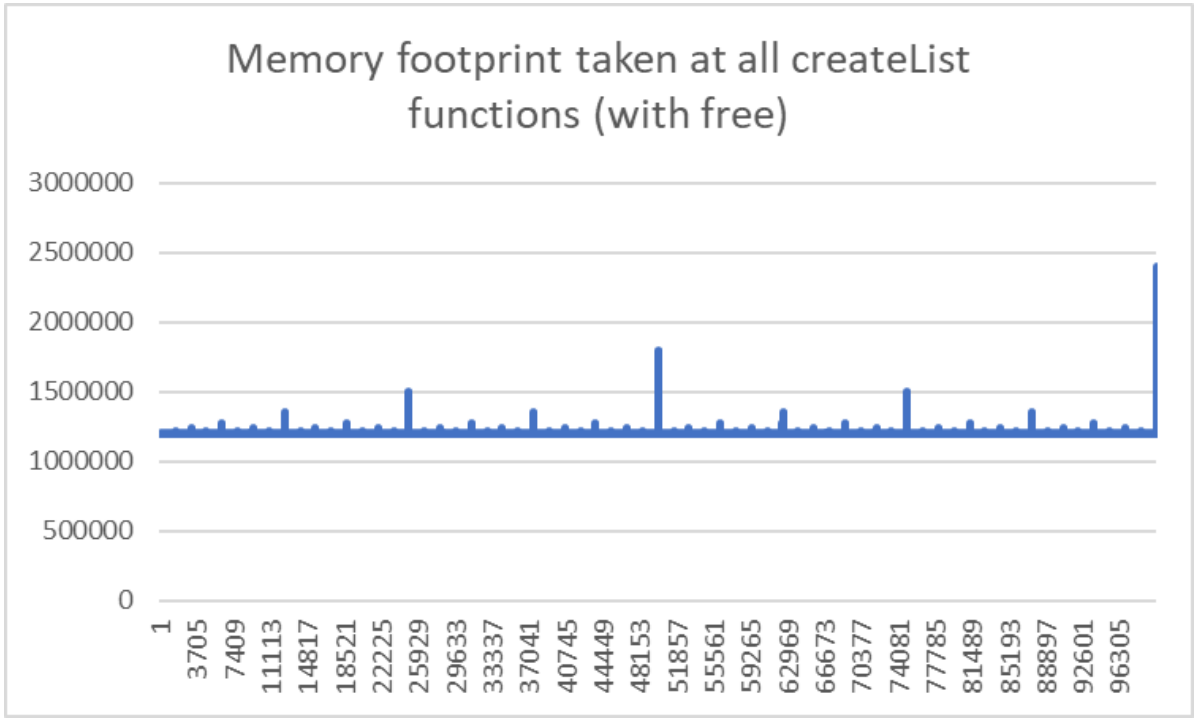
```
The mem-footprint (averaged over multiple rounds):
With free: 2400000,
Without free: 20027136
```
The difference (above two values): 17627136
The ratio (above two values): 8.345

```
The time taken (averaged over multiple rounds):
```
With freeElem:
real 0m12.099s
user 0m9.645s
sys 0m1.236s

Without freeElem:
real 0m30.896s
user 0m25.405s
sys 0m1.812s







Note: the values were overflowing in case of doing the second graph without **freeElem**

d. In what type of code structure will this performance be maximized, where will it be minimized? Why?

The code is designed keeping in mind a few but huge allocations and frees.

Too many allocations, or frees can make the page table big, and hence the **O(n)** lookup may no longer be sufficient. But we do have an advantage with cache locality, especially if we keep the array data structure sorted. For a few big allocations, and frees, the block reuse probability will also be higher.

Secondly, for such a situation we can use the library with minimal overhead, and no locks, and we can facilitate garbage collection.

e. Did you use locks in your library? Why or why not?

No, we didn't use locks in our library as no concurrent components exist in the code.