Test scenario 1:

Purpose:

Demonstrate the behavior of the four algorithms.

Assumption:

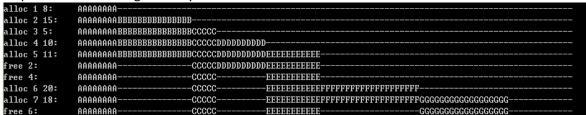
max_mem_size = 100, mem_start_address = 0

Test Case:

- 1. set up memory environment by following requests: alloc 1 8 -> alloc 2 15 -> alloc 3 5 -> alloc 4 10 -> alloc 5 11 -> free 2
 - -> free 4 -> alloc 6 20 -> alloc 7 18 -> free 6
- 2. submit request: alloc 8 8

Test Result:

Step 1 results following memory blocks:



Now there are four free blocks as follows, and last memory allocation occurs at 69.

block #	start address	size
1	8	15
2	28	10
3	49	20
4	87	13

Result of step 2:

First-fit: choose block 1. (search from 0, the first block larger than 8)



Next-fit: choose block 4. (search from 69, the first block larger than 8)



Best-fit: choose block 2. (the smallest block larger than 8)



Worst-fit: choose block 3. (the largest block larger than 8)

```
Alloc 8 8: AAAAAAAA------CCCCC-----EEEEEEEEEEHHHHHHHHH
```

Conclusion

- Best-fit: Choose the smallest free block of sufficient size.
- Worst-fit: Choose the largest free block of sufficient size.

- First-fit: Search from the beginning, choose the first free block of sufficient size.
- Next-fit: Search from where the last allocation occurs, choose the first free block of sufficient size.

Test scenario 2:

Purpose:

Compare the fragmentation resulted from best-fit and worst-fit.

Assumption:

max mem size = 100, mem start address = 0

Test Case:

- 1. Set up memory environment by following requests: alloc 1 14 -> alloc 2 6 -> alloc 3 16 -> free 2 -> alloc 4 12 -> alloc 5 20 -> free 4 -> alloc 6 14
- 2. Submit requests: alloc 7 3 -> alloc 8 8 -> alloc 9 5 -> alloc 10 6 -> alloc 11 5

Test Result:

Result of step 1:

Both best-fit and worst-fit produce following memory usage.

```
АААААААААА
alloc 1 14:
alloc 2 6:
          аааааааааааааВВВВВ-
alloc 3 16:
          AAAAAAAAAAAABBBBBBCCCCCCCCCCCCCCCC
free 2:
           AAAAAAAAAAAA-
                         -ccccccccccccc-
alloc 4 12:
          AAAAAAAAAAAA
                         CCCCCCCCCCCCCCDDDDDDDDDDDDDD
alloc 5 20:
          ааааааааааааа-
                         ree 4:
                                              AAAAAAAAAAAA-
                         -cccccccccccccc--
          AAAAAAAAAAAA
alloc 6 14:
                         -00000000000000000-
```

Now there are three free blocks as follows.

block #	start address	size
1	14	6
2	36	12
3	82	18

Result of step 2:

Fragments produced by best-fit: (3, 4, 2)

```
EEEEEEEEEEEEEEEEEFPPPPPPPPPPP
alloc 7 3:
             AAAAAAAAAAAAAGGG-
                              -ccccccccccccc-
alloc 8 8:
             AAAAAAAAAAAAAGGG-
                              -сссссссссссссниннинн
                                                       -EEEEEEEEEEEEEEEEEEEFFFFFFFFFFFFF
                                                       <u>-eeeeeeeeeeeeeeeeefppppppppppppfiiii</u>
alloc 9 5:
             AAAAAAAAAAAAGGG-
                              -сссссссссссссниннини
alloc 10 6:
             AAAAAAAAAAAAAAGGG-
                              -ссссссссссссснининин
                                                       <u>eeeeeeeeeeeeeeeeeffffffffffffffiiiijjjjj</u>
alloc 11 5:
             AAAAAAAAAAAAAGGG --- CCCCCCCCCCCCCCCHHHHHHHH
```

Fragments produced by worst-fit: (6, 1, 2)

```
alloc 7 3:
         AAAAAAAAAAAA
                                      EEEEEEEEEEEEEEEEEEEEFFFFFFFFFFFGGC
alloc 8 8:
         AAAAAAAAAAAAA
                     -cccccccccccccc-
                                      alloc 9 5:
        ааааааааааааа.
                     CCCCCCCCCCCCCIIIII-
                                      alloc 10 6:
         АААААААААА
                     CCCCCCCCCCCCCCIIIIIJJJJJ. - EEEEEEEEEEEEEEEEEEFFFFFFFFFFFFFFGGGHHHHHHHKKKKK
alloc 11 5:
        AAAAAAAAAAAA
```

Conclusion

- Best-fit: All remaining holes are too small to be useful.
- Worst-fit: Compared to best fit, increases the possibility that the remaining space can be used by another process.

Test scenario 3:

Purpose:

Compare first-fit with next-fit.

Assumption:

max_mem_size = 100, mem_start_address = 0

Test Case:

- 1. Set up memory environment by following requests: alloc 1 12 -> alloc 2 4 -> alloc 3 8 -> alloc 4 15 -> alloc 5 24 -> free 4
- 2. Submit requests: alloc 6 10 -> alloc 7 11 -> free 1 -> alloc 8 6 -> alloc 9 4 -> free 2 -> alloc 10 3

Test Result:

Result of step 1:

Both first-fit and next-fit produce following memory usage.

Result of step 2:

Memory holes produced by first-fit:

```
alloc 6 10:
alloc 7 11:
free 1:
             AAAAAAAAAABBBBCCCCCCCFFFFFFFFF
                                               AAAAAAAAAABBBBCCCCCCCFFFFFFFFF
                                               -FEFFFFFFFFFFFFFFFFFFFFGGGGGGGGGGG
                       -BBBBCCCCCCCFFFFFFFFFF
                                               alloc 8 6:
            нининн
                       -BBBBCCCCCCCCFFFFFFFFF
                                               -EEEEEEEEEEEEEEEEEEEEEEGGGGGGGGGG
alloc 9 4:
             HHHHHHIIIII--BBBBCCCCCCCFFFFFFFFFF
                                               EEEEEEEEEEEEEEEEEEEEEGGGGGGGGG
             нинини
                           CCCCCCCFFFFFFFFF
                                               EEEEEEEEEEEEEEEEEEEEEEGGGGGGGG
            НИНИННІІІЈЈЈ-
 lloc 10 3:
                          - CCCCCCCFFFFFFFFF
                                               EEEEEEEEEEEEEEEEEEEEEEGGGGGGGGG
```

Memory holes produced by next-fit:

Conclusion:

- First-fit: Tends to fragment the low memory area, leaving larger free block(s) in the end of the memory.
- Next-fit: Compared to first-fit, tends to chew up the end of the memory.