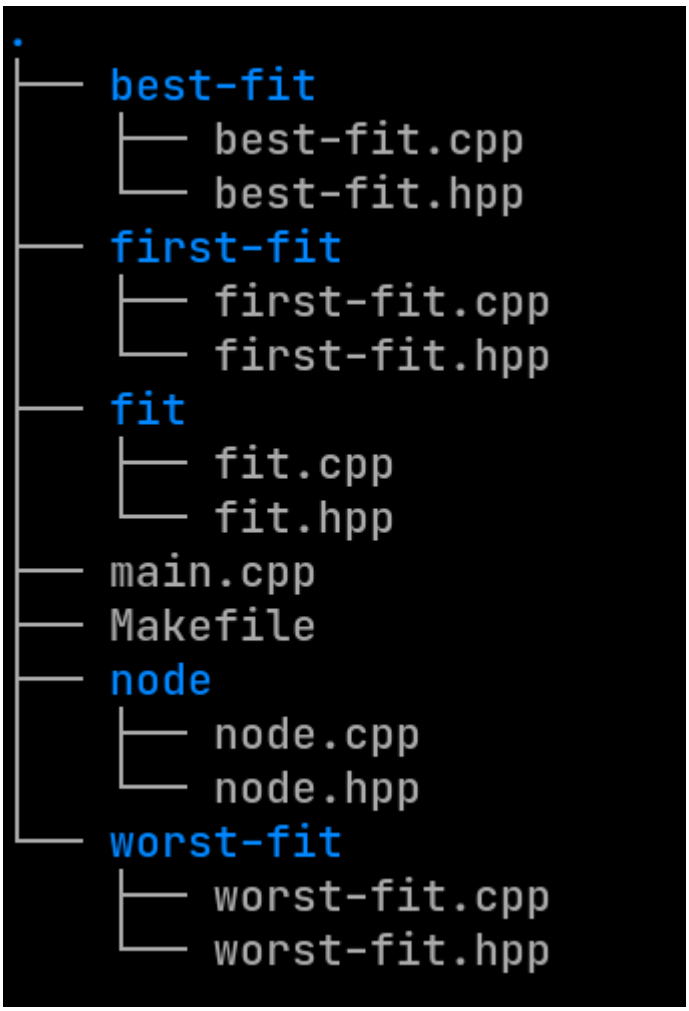
OS LAB 6

U18CO021: SAHIL BONDRE

1. To implement first fit, best fit and worst fit storage allocation algorithms for memory management.

Folder Structure:



node.hpp

```
#pragma once

class Node {
 public:
  int start;
  int end;
  int size;
  int id;
  Node(int start, int end, int id);
};
```

node.cpp

```
#include "node.hpp"

Node::Node(int start, int end, int id)
    : start(start), end(end), size(end - start + 1), id(id) {}
```

fit.hpp

```
#include <list>
#include <string>
#include <unordered_set>
#include <vector>

#include "../node/node.hpp"
#pragma once

using namespace std;

class Fit {
public:
  list<Node> empty_list;
  list<Node> filled_list;
  unordered_set<int> processes_ids;
  int memory_size;
  Fit(int memory_size);
  vector<string> print_lists();
  bool remove(int pid);
  virtual bool append(int process_size, int pid) = 0;

protected:
```

```
  void merge_list(list<Node> &list);
  void print_node(Node node, bool is_empty, vector<string> &res);
};
```

fit.cpp

```
#include "fit.hpp"

#include <iostream>
#include <list>
#include <vector>
#include <string>

#include "stdio.h"

using namespace std;

void Fit::merge_list(list<Node> &list) {
  if (list.empty())
    return;
  auto it = list.begin();
  ++it;
  while (it != list.end()) {
    auto prev = --it;
    ++it;
    if (prev->end + 1 == it->start) {
      // delete
      prev->end += it->size;
      prev->size += it->size;
      list.erase(it);
      it = prev;
    } else {
      ++it;
    }
  }
}

Fit::Fit(int memory_size) : memory_size(memory_size) {
  Node node(0, memory_size - 1, -1);
  empty_list.push_back(node);
}

void Fit::print_node(Node node, bool is_empty, vector<string> &res) {
  for (int i = node.start; i <= node.end; ++i) {
```

```
    if (is_empty) {
      char buff[100];
      snprintf(buff, sizeof(buff), "|%.2d   |", i);
      string s = buff;
      res.push_back(s);
    } else {
      char buff[100];
      snprintf(buff, sizeof(buff), "|%.2d \033[1;32m%.2d\033[0m|", i,
node.id);
      string s = buff;
      res.push_back(s);
    }
  }
}

vector<string> Fit::print_lists() {
  vector<string> res;
  auto it1 = empty_list.begin();
  auto it2 = filled_list.begin();

  while (it1 != empty_list.end() && it2 != filled_list.end()) {
    if (it1->start < it2->start) {
      print_node(*it1, true, res);
      ++it1;
    } else {
      print_node(*it2, false, res);
      ++it2;
    }
  }

  while (it1 != empty_list.end()) {
    print_node(*it1, true, res);
    ++it1;
  }

  while (it2 != filled_list.end()) {
    print_node(*it2, false, res);
    ++it2;
  }
  return res;
}

bool Fit::remove(int pid) {
  if (processes_ids.find(pid) == processes_ids.end()) {
    return false;
  }
```

```
  processes_ids.erase(pid);
  auto it = filled_list.begin();
  while (it != filled_list.end() && it->id != pid) {
    ++it;
  }

  int start = it->start;
  int size = it->size;
  int end = it->end;

  filled_list.erase(it);

  Node empty_node(start, end, -1);

  it = empty_list.begin();
  while (it != empty_list.end() && it->start <= start) {
    ++it;
  }
  empty_list.insert(it, empty_node);
  merge_list(empty_list);
  return true;
}
```

first-fit.hpp

```
#include <list>

#include "../fit/fit.hpp"
#include "../node/node.hpp"
#pragma once
using namespace std;

class FirstFit : public Fit {
 public:
  using Fit::Fit;
  bool append(int process_size, int pid);
};
```

first-fit.cpp

```
#include "first-fit.hpp"
```

```
bool FirstFit::append(int process_size, int pid) {
  // returns true if new_node could be inserted
  processes_ids.insert(pid);
  for (auto i = empty_list.begin(); i != empty_list.end(); ++i) {
    if (i->size >= process_size) {
      // fill here
      // find process after it
      int empty_node_end = i->end;
      int empty_node_start = i->start;
      auto filled_node = filled_list.begin();
      while (filled_node != filled_list.end() &&
             filled_node->end <= empty_node_end) {
        ++filled_node;
      }
      Node process(empty_node_start, empty_node_start + process_size -
1, pid);
      filled_list.insert(filled_node, process);
      i->start += process_size;
      i->size -= process_size;
      if (i->size == 0) {
        empty_list.erase(i);
      }
      return true;
    }
  }

  return false;
}
```

best-fit.hpp

```
#include <list>

#include "../fit/fit.hpp"
#include "../node/node.hpp"
#pragma once
using namespace std;

class BestFit : public Fit {
 public:
  using Fit::Fit;
  bool append(int process_size, int pid);
};
```

best-fit.cpp

```
#include "best-fit.hpp"

#include <limits.h>

bool BestFit::append(int process_size, int pid) {
  // returns true if new_node could be inserted
  list<Node>::iterator best_node_it = empty_list.begin();
  int best_node_size = INT_MAX;

  for (auto i = empty_list.begin(); i != empty_list.end(); ++i) {
    if (i->size <= best_node_size && i->size >= process_size) {
      best_node_size = i->size;
      best_node_it = i;
    }
  }

  if (best_node_size >= process_size && best_node_size != INT_MAX) {
    // fill here
    // find process after it
    int empty_node_end = best_node_it->end;
    int empty_node_start = best_node_it->start;
    auto filled_node = filled_list.begin();
    while (filled_node != filled_list.end() &&
           filled_node->end <= empty_node_end) {
      ++filled_node;
    }
    Node process(empty_node_start, empty_node_start + process_size - 1,
pid);
    filled_list.insert(filled_node, process);
    best_node_it->start += process_size;
    best_node_it->size -= process_size;
    if (best_node_it->size == 0) {
      empty_list.erase(best_node_it);
    }
    processes_ids.insert(pid);
    return true;
  }

  return false;
}
```

worst-fit.hpp

```
#include <list>

#include "../fit/fit.hpp"
#include "../node/node.hpp"
#pragma once
using namespace std;

class WorstFit : public Fit {
 public:
  using Fit::Fit;
  bool append(int process_size, int pid);
};
```

worst-fit.cpp

```
#include "worst-fit.hpp"

#include <limits.h>

bool WorstFit::append(int process_size, int pid) {
  // returns true if new_node could be inserted
  list<Node>::iterator best_node_it = empty_list.begin();
  int best_node_size = -1;

  for (auto i = empty_list.begin(); i != empty_list.end(); ++i) {
    if (i->size >= best_node_size && i->size >= process_size) {
      best_node_size = i->size;
      best_node_it = i;
    }
  }

  if (best_node_size >= process_size && best_node_size != -1) {
    // fill here
    // find process after it
    int empty_node_end = best_node_it->end;
    int empty_node_start = best_node_it->start;
    auto filled_node = filled_list.begin();
    while (filled_node != filled_list.end() &&
           filled_node->end <= empty_node_end) {
      ++filled_node;
    }
    Node process(empty_node_start, empty_node_start + process_size - 1,
```

```
pid);
    filled_list.insert(filled_node, process);
    best_node_it->start += process_size;
    best_node_it->size -= process_size;
    if (best_node_it->size == 0) {
      empty_list.erase(best_node_it);
    }
    processes_ids.insert(pid);
    return true;
  }

  return false;
}
```

main.cpp

```
#include <stdio.h>

#include <iostream>
#include <list>
#include <unordered_set>
#include <vector>

#include "best-fit/best-fit.hpp"
#include "first-fit/first-fit.hpp"
#include "node/node.hpp"
#include "worst-fit/worst-fit.hpp"

using namespace std;

const int MEMORY_SIZE = 16;

const string RED_PREFIX = "\033[1;31m";
const string RED_POSTFIX = "\033[0m";

const string BLUE_PREFIX = "\033[1;33m";
const string BLUE_POSTFIX = "\033[0m";

list<Node> first_fit_empty_list;
list<Node> first_fit_filled_list;

int main(int argc, char const *argv[]) {
  FirstFit ff(MEMORY_SIZE);
  BestFit bf(MEMORY_SIZE);
```

```
  WorstFit wf(MEMORY_SIZE);

  int first_pid = 1;
  int best_pid = 1;
  int worst_pid = 1;
  while (true) {
    string code;
    while (true) {
      cout << "Enter p to print, q to quit\n"
              "aa to add to all, af to add to first-fit, ab to add to "
              "best-fit,\n"
              "ra to remove from all, af to remove from first-fit, rb to
"
              "remove from best-fit\n: ";
      cin >> code;

      if (code == "p") {
        break;
      } else if (code[0] == 'a' && code.length() >= 2) {
        cout << "Enter process size: ";
        int n;
        cin >> n;
        char c;
        if (code[1] == 'f' || code[1] == 'a')
          if (ff.append(n, first_pid)) {
            cout << "FF New process created: Process " << first_pid <<
"\n";
            ++first_pid;
          } else {
            cout << RED_PREFIX + "FF Error: Not enough space\n" +
RED_POSTFIX;
          }

        if (code[1] == 'b' || code[1] == 'a')
          if (bf.append(n, best_pid)) {
            cout << "BF New process created: Process " << best_pid <<
"\n";
            ++best_pid;
          } else {
            cout << RED_PREFIX + "BF Error: Not enough space\n" +
RED_POSTFIX;
          }

        if (code[1] == 'w' || code[1] == 'a')
          if (wf.append(n, worst_pid)) {
            cout << "WF New process created: Process " << worst_pid <<
```

```
"\n";
            ++worst_pid;
          } else {
            cout << RED_PREFIX + "WF Error: Not enough space\n" +
RED_POSTFIX;
          }
      } else if (code[0] == 'r' && code.length() >= 2) {
        cout << "Enter process id: ";
        int id;
        cin >> id;
        if (code[1] == 'f' || code[1] == 'a')
          if (!ff.remove(id)) {
            cout << RED_PREFIX + "FF Error: Invalid process Ids\n" +
                        RED_POSTFIX;
          } else {
            cout << "Process removed\n";
          }

        if (code[1] == 'b' || code[1] == 'a')
          if (!bf.remove(id)) {
            cout << RED_PREFIX + "BF Error: Invalid process Ids\n" +
                        RED_POSTFIX;
          } else {
            cout << "Process removed\n";
          }

        if (code[1] == 'w' || code[1] == 'a')
          if (!wf.remove(id)) {
            cout << RED_PREFIX + "WF Error: Invalid process Ids\n" +
                        RED_POSTFIX;
          } else {
            cout << "Process removed\n";
          }
      } else if (code == "q") {
        exit(EXIT_SUCCESS);
      } else {
        cout << RED_PREFIX + "Error: Invalid Code\n" + RED_POSTFIX;
      }
    }

    cout << BLUE_PREFIX;
    printf("\n%s    %s    %s\n", "First First", "Best Fit", "Worst
Fit");
    cout << BLUE_POSTFIX;
    auto a = ff.print_lists();
    auto b = bf.print_lists();
```

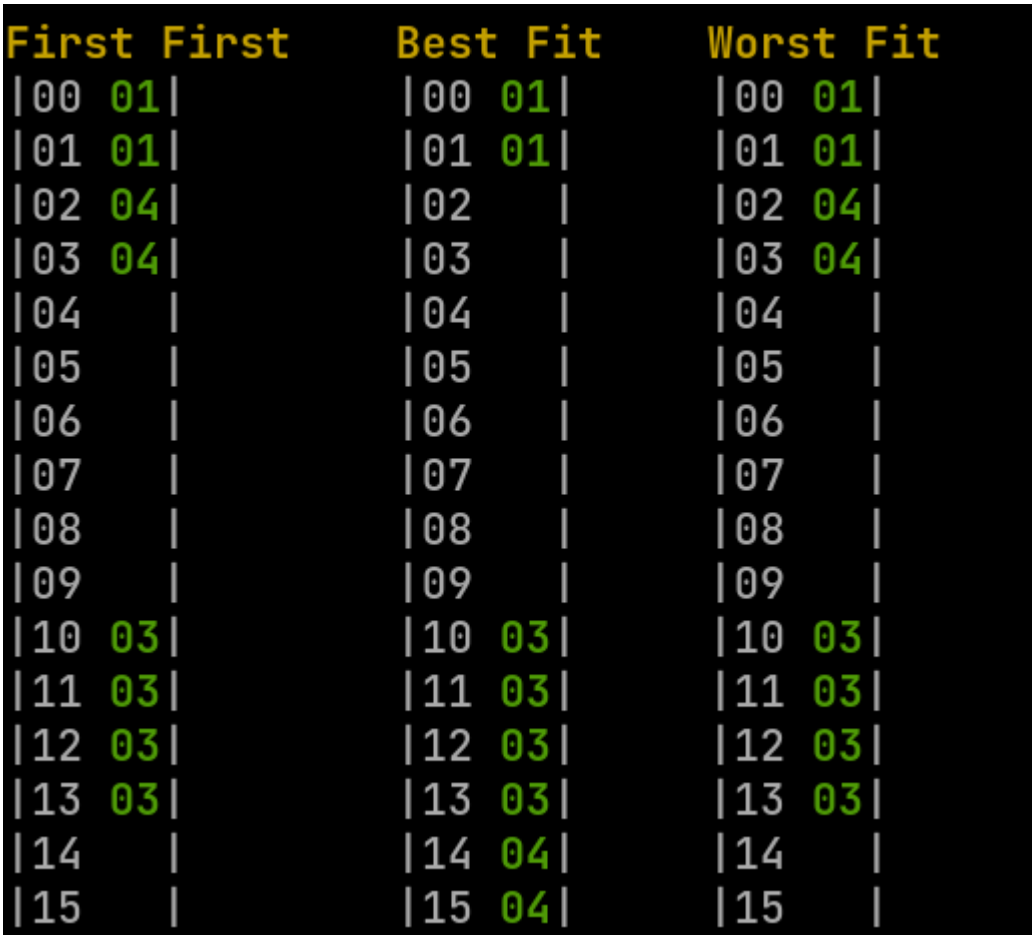
```
    auto c = wf.print_lists();
    for (int i = 0; i < a.size(); ++i) {
      cout << a[i] << "        " << b[i] << "     " << c[i] << "\n";
    }
  }
  return 0;
}
```

Makefile

```
CC=g++
cpp_formatter=clang-format
cpp_formatter_options=-style=Google -i --verbose
file_pattern='.*\.\(cpp\|hpp\|cc\|cxx\)'

all: main.cpp node/node.o fit/fit.o first-fit/first-fit.o
worst-fit/worst-fit.o best-fit/best-fit.o
	$(CC) ./main.cpp node/node.o fit/fit.o first-fit/first-fit.o
worst-fit/worst-fit.o best-fit/best-fit.o


%.o : %.cpp %.hpp
	$(CC) -c $< -o $@

format:
	find . -regex $(file_pattern) -exec $(cpp_formatter)
$(cpp_formatter_options) {} \;

.PHONY: clean
clean:
	rm ./*/*.o ./a.out
```

```
First Fit    Best Fit    Worst Fit
|00   |      |00   |     |00   |
|01   |      |01   |     |01   |
|02   |      |02   |     |02   |
|03   |      |03   |     |03   |
|04   |      |04   |     |04   |
|05   |      |05   |     |05   |
|06   |      |06   |     |06   |
|07   |      |07   |     |07   |
|08   |      |08   |     |08   |
|09   |      |09   |     |09   |
|10   |      |10   |     |10   |
|11   |      |11   |     |11   |
|12   |      |12   |     |12   |
|13   |      |13   |     |13   |
|14   |      |14   |     |14   |
|15   |      |15   |     |15   |
Enter p to print, q to quit
aa to add to all, af to add to first-fit, ab to add to best-fit,
ra to remove from all, af to remove from first-fit, rb to remove from best-fit
: 
```

```
First Fit    Best Fit    Worst Fit
|00 01|      |00 01|     |00 01|
|01 01|      |01 01|     |01 01|
|02   |      |02   |     |02   |
|03   |      |03   |     |03   |
|04   |      |04   |     |04   |
|05   |      |05   |     |05   |
|06   |      |06   |     |06   |
|07   |      |07   |     |07   |
|08   |      |08   |     |08   |
|09   |      |09   |     |09   |
|10 03|      |10 03|     |10 03|
|11 03|      |11 03|     |11 03|
|12 03|      |12 03|     |12 03|
|13 03|      |13 03|     |13 03|
|14   |      |14   |     |14   |
|15   |      |15   |     |15   |
Enter p to print, q to quit
aa to add to all, af to add to first-fit, ab to add to best-fit,
ra to remove from all, af to remove from first-fit, rb to remove from best-fit
: 
```



2. Write a program that implements the following Page replacement algorithm. i) LRU (Least Recently Used)ii) Optimal Page Replacement algorithm

main.cpp

```cpp
#include <algorithm>
#include <iomanip>
#include <iostream>
#include <sstream>
#include <stdlib.h>
#include <string>
#include <unordered_map>
#include <vector>

using namespace std;
int frames = 3;

// 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
```

```cpp
bool cmp(pair<string, int> &a, pair<string, int> &b) {
  return a.second < b.second;
}

int foundPair(vector<pair<string, int>> vec, string x) {
  for (int i = 0; i < vec.size(); ++i) {
    if (vec[i].first == x) {
      return i;
    }
  }
  return -1;
}

int findFutureIdx(vector<string> v, string x, int i) {
  for (int j = i; j < v.size(); ++j) {
    if (v[j] == x)
      return j;
  }
  return INT8_MAX;
}

void print(vector<pair<vector<string>, int>> output, int pageFault) {
  printf("\x1B[33mPage Allocation\033[0m\n");
  for (auto j : output[0].first)
    cout << j << endl;
  cout << "Page Fault: " << to_string(output[0].second) << endl;
  cout << endl;

  for (int i = 1; i < output.size(); ++i) {
    auto section = output[i];

    printf("\x1B[33mPage Allocation\033[0m\n");
    for (auto j : section.first)
      cout << j << endl;

    if (section.second != output[i - 1].second) {
      cout << "Page Fault: " << to_string(section.second) << endl;
    } else {
      cout << "No Page Fault Occured" << endl;
    }
    cout << endl;
  }

  cout << "Total Page Faults: " << to_string(pageFault) << endl;
  cout << endl;
```

```
}

int OptimalPageReplacement(string refString) {
  vector<string> page = {};
  istringstream iss(refString);
  string item;
  while (getline(iss, item, ' ')) {
    page.push_back(item);
  }

  int count = 0, pageFault = 0;
  vector<string> res(frames);
  vector<pair<vector<string>, int>> output;
  for (string x : page) {
    if (count < 3) {
      res[count] = x;
      ++pageFault;
    } else {
      if (find(res.begin(), res.end(), x) == res.end()) {
        ++pageFault;
        int futureIdx1 = findFutureIdx(page, res[0], count);
        int futureIdx2 = findFutureIdx(page, res[1], count);
        int futureIdx3 = findFutureIdx(page, res[2], count);

        int maxIndex = max(futureIdx1, max(futureIdx2, futureIdx3));
        if (maxIndex == futureIdx1) {
          res[0] = x;
        } else if (maxIndex == futureIdx2) {
          res[1] = x;
        } else if (maxIndex == futureIdx3) {
          res[2] = x;
        }
      }
    }

    output.push_back({res, pageFault});
    ++count;
  }
  print(output, pageFault);
  return pageFault;
}

int LRUPageReplacement(string refString) {
  vector<string> page = {};
  istringstream iss(refString);
  string item;
```

```
  while (getline(iss, item, ' ')) {
    page.push_back(item);
  }

  int count = 0, timer = 0, pageFault = 0;
  vector<string> res(frames);
  vector<pair<vector<string>, int>> output;
  vector<pair<string, int>> timerVector = {};

  for (string x : page) {
    if (!timerVector.empty())
      sort(timerVector.begin(), timerVector.end(), cmp);

    if (count < 3) {
      timerVector.push_back({x, timer});
      res[count] = x;
      ++pageFault;
    } else {
      // try to find if the page is there in map already
      // if there update the timer value only
      // if not there, take the least used and then insert the new page
there
      if (foundPair(timerVector, x) != -1) {
        // it is present in the map
        int idx = foundPair(timerVector, x);
        timerVector[idx].second = timer;
      } else {
        auto itr = timerVector.begin();
        auto findPage = find(res.begin(), res.end(), itr->first);
        int idx = findPage - res.begin();

        res[idx] = x;
        timerVector.erase(itr);
        timerVector.push_back({x, timer});
        ++pageFault;
      }
    }

    output.push_back({res, pageFault});
    ++timer;
    ++count;
  }
  print(output, pageFault);
  return pageFault;
}
```

```
int main() {
  cout << "Enter the page reference string: ";
  string refString;
  getline(cin, refString);

  printf("\x1B[32mLRU Page Replacement\033[0m\n");
  int pageFaultLRU = LRUPageReplacement(refString);

  cout << endl;
  printf("\x1B[32mOptimal Page Replacement\033[0m\n");
  int pageFaultOptimal = OptimalPageReplacement(refString);

  cout << '|' << setw(10) << " LRU Page Faults " << '|' << setw(10)
       << " Optimal Page Faults " << '|' << endl;
  cout << '|' << setw(17) << pageFaultLRU << '|' << setw(21) <<
pageFaultOptimal
       << '|' << endl;
  cout << endl;
  return 0;
}
```

```
Enter the page reference string: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
LRU Page Replacement
Page Allocation
7


Page Fault: 1

Page Allocation
7
0

Page Fault: 2

Page Allocation
7
0
1
Page Fault: 3

Page Allocation
2
0
1
Page Fault: 4

Page Allocation
2
0
1
No Page Fault Occured

Page Allocation
2
0
3
Page Fault: 5
```

```
Page Allocation
1
0
2
Page Fault: 11

Page Allocation
1
0
2
No Page Fault Occured

Page Allocation
1
0
7
Page Fault: 12

Page Allocation
1
0
7
No Page Fault Occured

Page Allocation
1
0
7
No Page Fault Occured

Total Page Faults: 12
```

```
Optimal Page Replacement
Page Allocation
7


Page Fault: 1

Page Allocation
7
0

Page Fault: 2

Page Allocation
7
0
1
Page Fault: 3

Page Allocation
2
0
1
Page Fault: 4

Page Allocation
2
0
1
No Page Fault Occured

Page Allocation
2
0
3
Page Fault: 5
```

```
Page Allocation
2
0
1
No Page Fault Occured

Page Allocation
2
0
1
No Page Fault Occured

Page Allocation
7
0
1
Page Fault: 9

Page Allocation
7
0
1
No Page Fault Occured

Page Allocation
7
0
1
No Page Fault Occured

Total Page Faults: 9

| LRU Page Faults | Optimal Page Faults |
|              12|                    9|
```