

**Lab report**

|  |  |
| --- | --- |
| **Course**: | Class Libraries and Data Structures |
| **Semester**: | 1st semester of the academic year **2021-2022** |
| **Major**: | Software Engineering |
| **Class**: | 2020 |
| **Student Name**: | 温长锟 |
| **Student ID:** | 222020321062106 |
| **Teacher:** | ZHAO, Hengjun (赵恒军) |

**School of Computer and Information Science**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | | Time Complexity and Runtime Analysis | | | |
| Date | | Oct 11，2021 | Type | | √ Confirmatory  √ Design  □Comprehensive |
| 1. **Objective & Requirements**    1. Understand the theoretical time complexity of an algorithm and know how to analyze it    2. Grasp the use of random numbers and techniques for measuring execution time in C++    3. Grasp runtime analysis of programs to show the effect of theoretical complexity on the time cost of real programs by running programs and measuring the time cost | | | | | |
| 1. **Experimental environment (**platform and software**)**   Windows 7 (or higher versions) + Visual Studio 2010 (or higher versions) | | | | | |
| 1. **Experimental content and design** (Main Content, Procedure, Codes and Results)   Task 1   1. You are provided with a template container based on singly linked list. Please read the source code and implement a new method AddTail() that can add a new element at the end of the linked list. 2. You are provided with a Company and Employee class. Please implement two methods for the Company class i.e.    * + - void inputEmployeeHead(int n);        - void inputEmployeeTail(int n);   The integer n is the method argument that specifies the  total number of employee to input. inputEmployeeHead() is  based on AddHead() and inputEmployeeTail() is based on  AddTail(). For each new employee, its name is of format  “Employee+ID”, e.g. “Employee123”, and its gross pay  is a randomly generated integer.   1. Using runtime analysis to measure the time costs of inputEmployeeHead and inputEmployeeTail by increasing the total number of employee n, e.g. n =1000, 2000, …, 10000, 20000, …, 100000, and so on. Record the time costs of the two methods for each n. Plot the data in a figure and try to fit the data using a curve (数据拟合，曲线拟合)。 2. Analyze the theoretical time complexity of inputEmployeeHead and inputEmployeeTail. Compare your theoretical analysis to the experimental data you obtained.   Step1.  read the source code and implement a new method AddTail() that can add a new element at the end of the linked list.    Step2：  implement two methods for the Company class i.e.   * + - * void inputEmployeeHead(int n);       * void inputEmployeeTail(int n);   The integer n is the method argument that specifies the  total number of employee to input. inputEmployeeHead() is  based on AddHead() and inputEmployeeTail() is based on  AddTail(). For each new employee, its name is of format  “Employee+ID”, e.g. “Employee123”, and its gross pay  is a randomly generated integer.    Step3：  Using runtime analysis to measure the time costs of inputEmployeeHead and inputEmployeeTail by increasing the total number of employee n, e.g. n =1000, 2000, …, 10000, 20000, …, 100000, and so on. Record the time costs of the two methods for each n.    Code:  #ifndef LISTTEMP\_H  #define LISTTEMP\_H  #define NULL 0  template<class T>  class ListTemp  {  private:  struct Node  {  T data;  Node\* next;  };  Node\* head;  int size;  public:  ListTemp();  ~ListTemp();  int getLength() const;  bool isEmpty() const;  void AddHead(const T& newData);  void AddTail(const T& newData);  };  template<class T>  ListTemp<T>::ListTemp()  {  head = NULL;  size = 0;  }  template<class T>  ListTemp<T>::~ListTemp()  {  Node\* current = head;  Node\* temp = NULL;  while (current != NULL)  {  temp = current;  current = current->next;  delete temp;  }  }  template<class T>  int ListTemp<T>::getLength() const  {  return size;  }  template<class T>  bool ListTemp<T>::isEmpty() const  {  return size == 0;  }  template<class T>  void ListTemp<T>::AddHead(const T& newData)  {  Node\* temp = new Node;  temp->next = head;  temp->data = newData;  head = temp;  size++;  }  template<class T>  void ListTemp<T>::AddTail(const T& newData)  {  //please implement this  Node\* temp = new Node;  temp->data = newData;  Node\* ptr = head;  while ((ptr->next) != NULL)  ptr = ptr->next;  temp->next = NULL;  ptr->next = temp;  size++;  }  #endif  #include "company.h"  #include <iostream>  using namespace std;  void Company::inputEmployeeHead(int n)  {  //please implement this  for (int i = 0; i < n; i++)  {  srand((unsigned int)time(0));  string s = "Employee";  string res = s + to\_string(n);  int pay = rand();  Employee emp(res, pay);  container.AddHead(emp);  }  }  void Company::inputEmployeeTail(int n)  {  for (int i = 0; i < n; i++)  {  srand((unsigned int)time(0));  string s = "Employee";  string res = s + to\_string(n);  int pay = rand();  Employee emp(res, pay);  container.AddTail(emp);  }  }  #include "listTemp.h"  #include "company.h"  #include <ctime>  #include <iostream>  using namespace std;  int main()  {  clock\_t begin1, end1, time1, begin2, end2, time2;  for (int num = 1000; num < 100000; num += 1500)  {  Company cmp;  begin1 = clock();  cmp.inputEmployeeHead(num);  end1 = clock();  time1 = end1 - begin1;  begin2 = clock();  cmp.inputEmployeeTail(num);  end2 = clock();  time2 = end2 - begin2;  cout << time1 << endl;  cout << time2 << endl;  }  return 0;  }  Result:    HEAD:    Linear model Poly1:  f(x) = p1\*x + p2  Coefficients (with 95% confidence bounds):  p1 = 10.21 (9.848, 10.57)  p2 = -3.495 (-7.818, 0.8281)  Goodness of fit:  SSE: 353.1  R-square: 0.9949  Adjusted R-square: 0.9946  RMSE: 4.429  Tail:    Linear model Poly2:  f(x) = p1\*x^2 + p2\*x + p3  Coefficients (with 95% confidence bounds):  p1 = 30.39 (28.25, 32.54)  p2 = -119.2 (-165.5, -72.94)  p3 = 286.8 (75.83, 497.9)  Goodness of fit:  SSE: 3.072e+05  R-square: 0.9984  Adjusted R-square: 0.9982  RMSE: 134.4  Answer for Additional questions  If the new node is inserted into the tail of the current linked list, the time complexity is the same as that of the header insertion method. | | | | | |
| 1. **Result analysis and discussion**（Analysis of experimental results and summing up the harvest and the existing problems）   The head insertion method is used to establish a single linked list. The order of reading data is opposite to the order of elements in the generated linked list. The insertion time of each node is O (1). If the length of the single linked list is n, the total time complexity is O (n).  To establish a single linked list by head interpolation, you need to traverse the linked list. The insertion time of each node is O (n). If the length of the single linked list is n, the total time complexity is O(n²)  Under the guidance of Mr. Zhao, I understand the theoretical time complexity of the algorithm and know how to analyze it. In this experiment, I successfully analyzed the time complexity of the two methods.  In addition, I also master the techniques of random numbers and measuring execution time in C + +. In this experiment, I used this technology to complete the time complexity analysis of head interpolation and tail interpolation.  Therefore, I master the runtime analysis of the program. After the experiment, I have been able to show the impact of theoretical complexity on the time cost of actual programs by running programs and measuring time cost. | | | | | |
| Comments & Evaluation | Content & Design (A-E) | | |  | |
| Procedure & Codes (A-E) | | |  | |
| Results (A-E) | | |  | |
| Analysis & Discussion (A-E) | | |  | |
| Score (A-E):  Feedback comments: | | | | |