Luis Anton P. Imperial	S-CSPC315	Monday, September 9, 2024
BCS32	Algorithms & Complexity	240909 – Mini Activity

Compute the time complexity of the given numbers, identify the correct complexity and show your solutions and answer. Failure to will not gain any mark on this activity. Screen shot your solution and answer, box your answer.

View raw file of answers at: https://dlsudphl-my.sharepoint.com/:u:/g/personal/ilp0824_dlsud_edu_ph/ERJ28WdOGHxFoi1B86dYmHEBNpInG7DB6oyeqj-HqCOrEQ?e=rfCht7

Table of Contents

Answers

In screenshot form, with final answers boxed in red

```
1  /*
2  De La Salle University – Dasmariñas
3  College of Information and Computer Studies
4  Computer Science Department
5
6  S–CSPC315: Algorithms and Complexity
7  240909 - Mini Activity
8  Monday, September 9, 2024
9
10 Mini Activity
11  Compute the time complexity of the given
   numbers, identify the correct complexity and show
    your solutions and answer. Failure to will not gain
   any mark on this activity. Screen shot your
   solution and answer, box your answer.
12
13 Luis Anton P. Imperial
14 BCS32
15
16 */
```

Luis Anton P. Imperial	S-CSPC315	Monday, September 9, 2024
BCS32	Algorithms & Complexity	240909 – Mini Activity

```
20
21  // 1.
22  int getFirstElement(int arr[], int size) {
23      return arr[0];     // --> O(1)
24  }
25  // Complexity: O(1)
26
27  // 2.
28  int binarySearch(int arr[], int l, int r, int x) {
29      while (l <= r) {
30          int mid = l + (r - l) / 2;
31              // (n/2)(n/4)
32              // --> log[sub(2)](n) = O(log n)
33          if (arr[mid] == x) return mid;
34              // O(1)
35          if (arr[mid] < x) l = mid + 1;
36              // O(1)
37          else r = mid - 1;
38              // O(1)
39      }
40      return -1;
41  }
42  // Complexity: O(log n)
```

```
44  // 3.
45  int findMax(int arr[], int size) {
46      int max = arr[0];                   // O(1)
47      for (int i = 1; i < size; i++) {    // O(n)
48          if (arr[i] > max) max = arr[i]; // O(1)
49      }
50      return max;
51  }
52  // Complexity: O(n)
53
54  // 4.
55  void merge(int arr[], int l, int m, int r) {
56      // Merge function for merge sort
57      // Details omitted for brevity
58  }
59  void mergeSort(int arr[], int l, int r) {
60      if (l < r) {
61          int m = l + (r - l) / 2;
62          // O(1)
63          mergeSort(arr, l, m);
64          mergeSort(arr, m + 1, r);
65          // O(log n)
66          merge(arr, l, m, r);
67          // O(n)
68      }
69  }
70  // Complexity:
71  //     O(log n) * O(n)
72  //   = O(n log n)
```

```
74  // 5.
75  void bubbleSort(int arr[], int size) {
76      for (int i = 0; i < size - 1; i++) {
77                          // --> O(n^2) * O(n) = O(n^3)
78          for (int j = 0; j < size - i - 1; j++) {
79                          // --> O(n) * O(n) = O(n^2)
80              if (arr[j] > arr[j + 1]) {
81                          // --> O(n)
82                  swap(arr[j], arr[j + 1]);
83                          // --> O(1)
84              }
85          }
86      }
87  }
88  // Complexity: O(n^3)
89
90  // 6.
91  void printAllTriplets(int arr[], int size) {
92      for (int i = 0; i < size; i++) {
93          //  O(n^2) * O(n) --> O(n^3)
94          for (int j = i + 1; j < size; j++) {
95              // O(n) * O(n) --> O(n^2)
96              for (int k = j + 1; k < size; k++) {
97                  // O(n)
98                  cout << arr[i] << ", " << arr[j] << ", " << arr[
    k] << endl;
99              }
100         }
101     }
102 }
103 // Complexity: O(n^2)
104
```

Luis Anton P. Imperial	S-CSPC315	Monday, September 9, 2024
BCS32	Algorithms & Complexity	240909 – Mini Activity

```
105 // 7.
106 void permute(string str, int l, int r) {
107     if (l == r) {              // --> O(1)
108         cout << str << endl;
109         } else {
110         for (int i = l; i <= r; i++) {
111           // --> O(n)
112             swap(str[l], str[i]); // --> O(1)
113             permute(str, l + 1, r);
114               // --> O(n!)
115             swap(str[l], str[i]); // Backtrack
116               // --> O(1)
117             }
118         }
119 }
120 // Complexity:
121 // O(1) * O(n) * O(n!) * O(1)
122 // = O(n * n!)
123
124 // 8.
125 int fib(int n) {
126     if (n <= 1) return n;
127     // --> O(1)
128     return fib(n - 1) + fib(n - 2);
129     // --> O(n) + O(n) = O(2^n)
130
131 }
132 // Complexity:
133 // = O(2^n)
```

```
134
135 // 9.
136 void heapify(int arr[], int n, int i) {// Heapify
    function for heap sort// Details omitted for brevity}
137
138 void heapSort(int arr[], int n) {
139     for (int i = n / 2 - 1; i >= 0; i--)
140         // --> O(n)
141         heapify(arr, n, i);     // --> O(log n)
142     for (int i = n - 1; i >= 0; i--) {
143         // --> O(n)
144         swap(arr[0], arr[i]);   // --> O(1)
145         heapify(arr, i, 0);     // --> O(log n)
146     }
147 }
148 // Complexity:
149 // O(n) * O(log n)
150 // = O(n log n)
151
152 // 10.
153     bool hasDuplicate(std::vector<int>& nums){
154         std::unordered_set<int> seen;
155     for (int num : nums) {
156         // --> O(n)
157         if (seen.find(num) != seen.end()) {
158             // --> O(1)
159             return true; // Duplicate found}
160         seen.insert(num);}
161     return false; // No duplicates
162     }
163 }
164 // Complexity: O(n)
165
```

```
166 // 11.
167     int countSetBits(int n) {
168      if (n == 0) return 0; // --> O(1)
169      return (n & 1) + countSetBits(n >> 1);
170      // --> O(1) + O(log n)
171
172 }
173 // Complexity: O(log n)
174
175
176 // 12.
177     void printPairs(int arr[], int size) {
178     for (int i = 0; i < size; i++) {
179                         // --> O(n) * O(n) = O(n^2)
180         for (int j = i + 1; j < size; j++) {
181                         // --> O(n)
182             cout << "(" << arr[i] << ", " << arr[j] << ")" <<
    endl;}}
183                         // --> O(1)
184
185 }
186 // Complexity: O(n^2)
187
188 // 13.
189     bool containsSubstring(const string& str, const
    string& pattern) {
190      size_t pos = str.find(pattern);    // O(n)
191     return pos != string::npos;         // O(1)
192
193 }
194 // Complexity: O(n)
```

```
// 14.
   void generateSubarrays(int arr[], int size) {
   for (int start = 0; start < size; start++) {
   // --> O(n²) * O(n) = O(n³)
      for (int end = start; end < size; end++) {
      // --> O(n) * O(n) = O(n²)
         for (int i = start; i <= end; i++) {
         // --> O(n)
            cout << arr[i] << " ";
            // --> O(1)
         }
         cout << endl;}}

}
// Complexity: O(n³)

// 15.
   void reverseArray(int arr[], int size) {
    int start = 0, end = size - 1; // --> O(1)
    while (start < end) {              // --> O(n)
       swap(arr[start], arr[end]);  // --> O(1)
       start++;                     // --> O(1)
       end--;                       // --> O(1)
    }
 }
// Complexity:
// O(1) * O(n) * O(1) * O(1) * O(1)
// O(n)
```

In raw form

// 1.

Luis Anton P. Imperial	S-CSPC315	Monday, September 9, 2024
BCS32	Algorithms & Complexity	240909 – Mini Activity

```
Int getFirstElement(int arr[], int size) {
    Return arr[0];    // → O(1)
 }
// Complexity: O(1)


// 2.
Int binarySearch(int arr[], int l, int r, int x) {
   While (l <= r) {
      Int mid = l + (r – l) / 2;
         // (n/2)(n/4)
         // → log[sub(2)](n) = O(log n)
      If (arr[mid] == x) return mid;
         // O(1)
      If (arr[mid] < x) l = mid + 1;
         // O(1)
      Else r = mid – 1;
         // O(1)
   }
   Return -1;
}
// Complexity: O(log n)


// 3.
Int findMax(int arr[], int size) {
   Int max = arr[0];              // O(1)
```

```
   For (int I = 1; I < size; i++) {    // O(n)
      If (arr[i] > max) max = arr[i]; // O(1)
   }
   Return max;
}
// Complexity: O(n)


// 4.
Void merge(int arr[], int l, int m, int r) {
    // Merge function for merge sort
    // Details omitted for brevity
 }
 Void mergeSort(int arr[], int l, int r) {
    If (l < r) {
       Int m = l + (r – l) / 2;
       // O(1)
       mergeSort(arr, l, m);
       mergeSort(arr, m + 1, r);
       // O(log n)
       Merge(arr, l, m, r);
       // O(n)
    }
 }
// Complexity:
//     O(log n) * O(n)
```

```
//    = O(n log n)


// 5.
Void bubbleSort(int arr[], int size) {
   For (int I = 0; I < size – 1; i++) {
                  // → O(n^2) * O(n) = O(n^3)
      For (int j = 0; j < size – I – 1; j++) {
                  // → O(n) * O(n) = O(n^2)
         If (arr[j] > arr[j + 1]) {
                  // → O(n)
            Swap(arr[j], arr[j + 1]);
                  // → O(1)
         }
      }
   }
}
// Complexity: O(n^3)


// 6.
Void printAllTriplets(int arr[], int size) {
   For (int I = 0; I < size; i++) {
      //  O(n^2) * O(n) → O(n^3)
      For (int j = I + 1; j < size; j++) {
         // O(n) * O(n) → O(n^2)
         For (int k = j + 1; k < size; k++) {
```

```
            // O(n)
            Cout << arr[i] << ", " << arr[j] << ", " << arr[k] << endl;
          }
        }
      }
}
// Complexity: O(n^2)


// 7.
Void permute(string str, int l, int r) {
   If (l == r) {            // → O(1)
       Cout << str << endl;
       } else {
       For (int I = l; I <= r; i++) {
         // → O(n)
          Swap(str[l], str[i]); // → O(1)
          Permute(str, l + 1, r);
            // → O(n!)
          Swap(str[l], str[i]); // Backtrack
            // → O(1)
          }
      }
}
// Complexity:
// O(1) * O(n) * O(n!) * O(1)
```

```
// = O(n * n!)


// 8.
Int fib(int n) {
    If (n <= 1) return n;
    // → O(1)
    Return fib(n – 1) + fib(n – 2);
    // → O(n) + O(n) = O(2^n)

}
// Complexity:
// = O(2^n)


// 9.
Void heapify(int arr[], int n, int i) {// Heapify function for heap sort// Details omitted for brevity}


Void heapSort(int arr[], int n) {
   For (int I = n / 2 – 1; I >= 0; i--)
      // → O(n)
      Heapify(arr, n, i);     // → O(log n)
   For (int I = n – 1; I >= 0; i--) {
      // → O(n)
      Swap(arr[0], arr[i]);   // → O(1)
      Heapify(arr, I, 0);     // → O(log n)
   }
```

```
}
// Complexity:
// O(n) * O(log n)
// = O(n log n)


// 10.
   Bool hasDuplicate(std::vector<int>& nums){
      Std::unordered_set<int> seen;
   For (int num : nums) {
      // → O(n)
      If (seen.find(num) != seen.end()) {
         // → O(1)
         Return true; // Duplicate found}
      Seen.insert(num);}
   Return false; // No duplicates
   }
}
// Complexity: O(n)


// 11.
   Int countSetBits(int n) {
    If (n == 0) return 0; // → O(1)
    Return (n & 1) + countSetBits(n >> 1);
    // → O(1) + O(log n)
```

```
}
// Complexity: O(log n)



// 12.
   Void printPairs(int arr[], int size) {
   For (int I = 0; I < size; i++) {
                  // → O(n) * O(n) = O(n^2)
      For (int j = I + 1; j < size; j++) {
                  // → O(n)
         Cout << "(" << arr[i] << ", " << arr[j] << ")" << endl;}}
                  // → O(1)


}
// Complexity: O(n^2)


// 13.
   Bool containsSubstring(const string& str, const string& pattern) {
    Size_t pos = str.find(pattern);    // O(n)
   Return pos != string::npos;        // O(1)


}
// Complexity: O(n)


// 14.
```

Luis Anton P. Imperial	S-CSPC315	Monday, September 9, 2024
BCS32	Algorithms & Complexity	240909 – Mini Activity

```
Void generateSubarrays(int arr[], int size) {
For (int start = 0; start < size; start++) {
// → O(n^2) * O(n) = O(n^3)
  For (int end = start; end < size; end++) {
  // → O(n) * O(n) = O(n^2)
    For (int I = start; I <= end; i++) {
    // → O(n)
      Cout << arr[i] << " ";
      // → O(1)
    }
    Cout << endl;}}

}
// Complexity: O(n^3)

// 15.
  Void reverseArray(int arr[], int size) {
   Int start = 0, end = size – 1; // → O(1)
   While (start < end) {          // → O(n)
     Swap(arr[start], arr[end]);  // → O(1)
     Start++;                     // → O(1)
     End--;                       // --> O(1)
   }
 }
// Complexity:
```

Luis Anton P. Imperial	S-CSPC315	Monday, September 9, 2024
BCS32	Algorithms & Complexity	240909 – Mini Activity

// O(1) * O(n) * O(1) * O(1) * O(1)

// O(n)