**Лабораторна робота № 1.**

***Тема:* “Аналіз часу виконання програми”.**

***Мета:*** Опанувати навичками визначення часу та порядку степеня росту алгоритму, набути навичок в розробці блок-схем, псевдокодів алгоритмів та написанні програм з використанням розгалужень та циклів.

***Теоретичні відомості:***

Поняття алгоритму, що належить до фундаментальних концепцій інформатики, виникло задовго до появи ЕОМ і стало одним з основних понять математики. Алгоритм Евкліда для знаходження найбільшого спільного дільника двох натуральних чисел, винайдений у III ст. до нашої ери, вважається найпершим відомим алгоритмом. Саме ж слово АЛГОРИТМ є перефразуванням географічної назви Хорезмської місцевості (тепер місто Хіва, Узбекистан) уродженцем якої був відомий узбецькій математик Мухаммед ібн Муса ал-Хорезмі (близько 825 р. нашої ери).

Під алгоритмом розуміють скінчену сукупність правил дій, які виконуються у певному порядку для розв’язання всіх задач даного класу.

**Властивості алгоритму:**

1. Масовість – можливість застосування до будь-яких вхідних даних задач виділеного класу;

2. Визначеність (детермінованість) – тобто набір вказівок має бути точний, не залежний від виконавця.

3. Дискретність – розчленованість процесу, що визначається алгоритмом, на окремі елементарні операції, можливість виконання яких людиною, або машиною не викликає сумнівів.

4. Зрозумілість – знання виконавця проте, що треба робити для виконання цього алгоритму.

5. Результативність – скінченність процесу перетворення інформації, тобто за певне число кроків повинен бути добутий шуканий результат, або доведено, що алгоритм не застосований до даної множини початкових даних.

Для більшості проблем існує багато різних алгоритмів. Який з них вибрати для вирішення конкретного завдання? Це питання дуже ретельно опрацьовується в програмуванні.

Ефективність програми (коду) є дуже важливою її характеристикою. Користувач завжди віддає перевагу ефективнішому рішенню навіть в тих випадках, коли ефективність не є вирішальним чинником.

Ефективність програми має дві складові: пам'ять (або простір) і час.

Просторова ефективність вимірюється кількістю пам'яті, потрібної для виконання програми.

Комп'ютери володіють обмеженим об'ємом пам'яті. Якщо дві програми реалізують ідентичні функції, то та, яка використовує менший об'єм пам'яті, характеризується більшою просторовою ефективністю. Іноді пам'ять стає домінуючим чинником в оцінці ефективності програм. Проте в останній роки у зв'язку з швидким її здешевленням ця складова ефективності поступово втрачає своє значення.

Часова ефективність програми визначається часом, необхідним для її виконання.

На час виконання програми мають вплив наступні фактори:

Введення вхідної інформації в програму.

Кількість скомпільованого коду програми, що виконується.

Машинні інструкції, які використовуються для виконання програми.

Часова складність алгоритму відповідної програми.

Час виконання програми можна визначити як функцію від вхідних даних.

Для опису швидкості росту функції використовується  – символіка. Тобто, коли ми говоримо, що час виконання  деякої програми має порядок , то розуміємо, що існують додатні  і  такі, що для всіх , більших або рівних , виконується нерівність . Усі функції часу виконання визначені на множині невід’ємних цілих чисел і їх значення також невід’ємні, але необов’язково цілі.

Будемо казати, що  має порядок , якщо існують константи  і  такі, що для всіх  виконується нерівність .

Для програм, у яких час виконання має порядок , говорять, що вони мають порядок росту . Коли використовують символіку , мають на увазі не точний час виконання, а лише верхню межу, або час виконання в найгіршому випадку, при чому з точністю до постійного множника.

 – функції виражають відносну швидкість алгоритму в залежності від деякої змінної.

Кращий спосіб порівняння ефективності алгоритмів полягає в зіставленні їх порядків складності. Цей метод застосовний як до часової, так і просторової складності. Порядок складності алгоритму виражає його ефективність звичайно через кількість оброблюваних даних.

Наприклад, деякий алгоритм може істотно залежати від розміру оброблюваного масиву. Якщо, скажімо, час обробки подвоюється з подвоєнням розміру масиву, то порядок часової складності алгоритму визначається як розмір масиву.

Порядок алгоритму - це функція, домінуюча над точним виразом часової складності.

Функція f(n) має порядок O(g(n)), якщо є константа К і лічильник n0, такі, що f(n)<=(K\*g(n), для n>n0.

O-функції виражають відносну швидкість алгоритму залежно від деякої змінної (або змінних).

Існують три правила для визначення складності:

1. ;

2. Правило добутку.

Якщо  і  мають степені росту  і  відповідно, то добуток  має степінь росту ; або O(f/g)=O(f)/O(g);

3. Правило сум.

Якщо  і  мають степені росту  і  відповідно, тоді час послідовного виконання фрагментів  і , тобто , має степінь росту .

Перше правило декларує, що постійні множники не мають значення для визначення порядку складності.  
О(1,5\*N)=O(N)

З другого правила виходить, що порядок складності добутку двох функцій рівний добутку їх складностей.

O((17\*N)\*N)= O(17\*N)\*O(N) = O(N)\*O(N)=O(N\*N) = O(N2)

З третього правила виходить, що порядок складності суми функцій визначається як порядок домінанти (максимуму) першого і другого доданків, тобто вибирається найбільший порядок.

O(N5+N2)=O(N5)

**O-складність алгоритмів:**

O(1)  
Більшість операцій в програмі виконуються тільки раз або тільки кілька разів алгоритмами константної складності. Будь-який алгоритм, що завжди вимагає незалежно від розміру даних одного і того ж часу, має константну складність.

О(N)  
Час роботи програми лінійний звичайно коли кожен елемент вхідних даних потрібно обробити лише лінійне число раз.

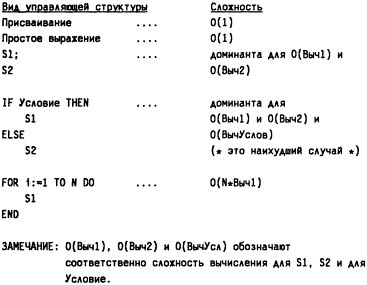
О(N2), О(N3), О(Nа)  
Поліноміальна складність. О(N2) -квадратична складність, О(N3) - кубічна складність

О(Log(N))  
Коли час роботи програми логарифмічний, програма починає працювати набагато повільніше із збільшенням N. Такий час роботи зустрічається звичайно в програмах, які ділять велику проблему на маленькі і вирішують їх окремо.

O(N\*log( N))  
Такий час роботи мають ті алгоритми, які ділять велику проблему на маленькі, а потім, вирішивши їх, поєднують їх рішення.

O(2N)  
Експоненціальна складність. Такі алгоритми найчастіше виникають в результаті підходу іменованого метод грубої сили.

Програміст повинен уміти проводити аналіз алгоритмів і визначати їх складність. Часова складність алгоритму може бути порахована виходячи з аналізу його керуючих структур.



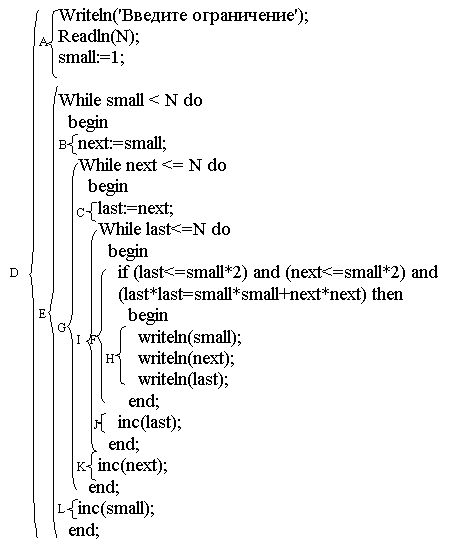
Алгоритми без циклів і рекурсивних викликів мають константну складність. Якщо немає рекурсії і циклів, всі управляючі структури можуть бути зведені до структур константної складності. Отже, і весь алгоритм також характеризується константною складністю.

Визначення складності алгоритму в основному зводиться до аналізу циклів і рекурсивних викликів.

Наприклад, розглянемо алгоритм обробки елементів масиву.  
For i:=1 to N do  
Begin  
...  
End;

Складність цього алгоритму O(N), оскільки тіло циклу виконується N раз, і складність тіла циклу рівна O(1).

Якщо один цикл вкладений в іншій і обидва цикли залежать від розміру однієї і тієї ж змінної, то вся конструкція характеризується квадратичною складністю.  
For i:=1 to N do  
For j:=1 to N do   
Begin  
...  
End;  
Складність цієї програми О(N2).



Давайте оцінимо складність програми "Трійки Піфагора"

Існують два способи аналізу складності алгоритму: висхідний (від внутрішніх керуючих структур до зовнішніх) і низхідний (від зовнішніх і внутрішнім).

;

;

;

;



Складність даного алгоритму  як правило 90% часу виконання роботи програми потребує виконання повторень і лише 10% складає безпосередньо обчислення. Аналіз складності програми показує на які фрагменти випадають ці 90% - це цикли найбільшої глибини вкладеності.

**Правила аналізу алгоритму**

Перед формулюванням загальних правил аналізу програм дозвольте нагадати, що знаходження точної верхньої межі часу виконання програм тільки в окремих випадках є простим, в загальному випадку ця задача є інтелектуальним викликом досліднику. Тому не існує вичерпної безлічі правил аналізу програм.

***Декілька правил аналізу програм.*** У загальному випадку час виконання оператора або групи операторів можна параметризувати за допомогою розміру вхідних даних і/або однієї або декількох змінних. Але для часу виконання програми в цілому допустимим параметром може бути тільки п, розмір вхідних даних.

1. Час виконання операторів привласнення, читання і запису звичайно має порядок 0(1). Є  
декілька виключень з цього правила, наприклад в мові PL/I, де можна привласнювати великі  
масиви, або в будь-яких інших мовах, що допускають виклики функцій в операторах  
привласнення.

2. Час виконання послідовності операторів визначається за допомогою правила сум.  
Тому ступінь зростання часу виконання послідовності операторів без визначення  
констант пропорційності співпадає з найбільшим часом виконання оператора в даній  
послідовності.

3. Час виконання умовних операторів складається з часу виконання умовно виконуваних  
операторів і часу обчислення самого логічного виразу. Час обчислення логічного  
виразу звичайно має порядок 0(1). Час для всієї конструкції if-then-else складається з  
часу обчислення логічного виразу і найбільшого з часів, необхідного для  
виконання операторів, виконуваних при значенні логічного виразу true (істина) і при  
значенні false (хиба).

4. Час виконання циклу є сумою часу всіх виконуваних ітерацій циклу, які в свою  
чергу складаються з часу виконання операторів тіла циклу і часу обчислення умови  
припинення циклу (звичайно останнє має порядок 0(1)). Часто час виконання циклу обчислюється, нехтуючи визначенням констант пропорційності, як добуток кількості виконаних ітерацій циклу на найбільший можливий час виконання операторів тіла циклу. Час виконання кожного циклу, якщо в програмі їх декілька, повинен визначатися окремо.

5. Виклики процедур

Для програм, що містять декілька процедур (серед яких немає рекурсивних), можна підрахувати загальний час виконання програми шляхом послідовного знаходження часу виконання процедур, починаючи з тієї, яка не має викликів інших процедур. Оскільки ми припустили, що всі процедури нерекурсивні, то повинна існувати хоча б одна процедура, що не має викликів інших процедур. Потім можна визначити час виконання процедур, що викликають цю процедуру, використовуючи вже обчислений час виконання процедури, що викликається. Продовжуючи цей процес, знайдемо час виконання всіх процедур і, нарешті, час виконання всієї програми.

6. Якщо є рекурсивні процедури, то не можна упорядкувати всі процедури так, щоб кожна процедура викликала тільки процедури, час виконання яких підрахований на попередньому етапі. В цьому випадку ми повинні з кожною рекурсивною процедурою пов'язати тимчасову функцію Т(п), де п визначає об'єм аргументів процедури. Потім ми повинні одержати рекуррентне співвідношення для Т(п), тобто рівняння (або нерівність) для Т(п), де беруть участь значення Т(к) для різних значень к.

Проаналізуємо просту рекурсивну програму.

Рекурсивна програма обчислення факторіалу

function fact(n: integer ): integer;

{ fact (n) обчислює n! } begin \

(1) if n <= 1 then

(2) fact:= 1 else

(3) fact:= n \*fact{n - 1)  
end; {fact)

Природною мірою об'єму вхідних даних для функції fact є значення п. Позначимо через Т(п) час виконання програми. Час виконання для рядків (1) і (2) має порядок 0(1), а для рядка (3) — 0(1)+ Т(п - 1). Таким чином, для деяких констант с і d маємо

Т(п)=с + Т(п -1), якщо п > 1,

Т(п)=d, якщо п < 1. (1.1)

Вважаючи, що п > 2, і розкриваючи відповідно до співвідношення (1.1) вираз Т(п - 1) (тобто підставляючи в (1.1) п - 1 замість п), одержимо Т(п)= 2с + Т(п - 2). Аналогічно, якщо п > 3, розкриваючи Т(п - 2), одержимо Т(п)= Зс + Т(п - 3). Продовжуючи цей процес, в загальному випадку для деякого i, п > i, маємо Т(п)= ic + Т(п - i). Поклавши в останньому виразі i — п - 1, остаточно одержуємо

Т(п)= с(п- 1)+ T(l)= с(n- l)+ d. (1.2)

З (1.2) витікає, що Т(п) має порядок 0(п). Відзначимо, що в даному прикладі аналізу програми ми припускали, що операція перемножування двох цілих чисел має порядок 0(1). На практиці, проте, дану програму не можна використовувати для обчислення факторіалу при великих значеннях п, оскільки розмір одержуваних в процесі обчислення цілих чисел може перевищувати довжину машинного слова.

Загальний метод рішення рекуррентних співвідношень, подібних співвідношенню з даного прикладу, полягає в послідовному розкритті виразів Т(к) в правій частині рівняння (шляхом підстановки в початкове співвідношення К замість п до тих пір, поки не вийде формула, у якої в правій частині відсутнє Т (як у формулі (1.2)). При цьому часто доводиться знаходити суми різних послідовностей; якщо значення таких сум не можна обчислити точно, то для сум знаходяться верхні межі, що дозволяє, у свою чергу, одержати верхні межі для Т(п).

7. Програми з операторами безумовного переходу

При аналізі часу виконання програм ми неявно припускали, що всі галуження в ході виконання процедур здійснювалися за допомогою умовних операторів і операторів циклів. Ми зупиняємося на цьому факті, оскільки визначали час виконання .великих груп операторів шляхом застосування правила сум до цих груп. Проте оператори безумовного переходу (такі як goto) можуть породжувати складнішу логічну групову структуру. У принципі, оператори безумовного переходу можна виключити з програми. Але, на жаль, мова Pascal не має операторів дострокового припинення циклів, тому оператори переходу часто використовуються в подібних ситуаціях.

Пропонується наступний підхід для роботи з операторами безумовного переходу, виконуючих вихід з циклів, який гарантує відстежування всього циклу. Оскільки достроковий вихід з циклу швидше за все здійснюється після виконання якої-небудь логічної умови, то ми можемо припустити, що ця умова ніколи не виконується. Але цей консервативний підхід ніколи не дозволить зменшити час виконання програми, так як ми припускаємо повне виконання циклу. Для програм, де достроковий вихід з циклу не виключення, а правило, консервативний підхід значно переоцінює ступінь зростання часу виконання в якнайгіршому випадку. Якщо ж перехід здійснюється до раніше виконаних операторів, то в цьому випадку взагалі не можна ігнорувати оператора безумовного переходу, оскільки такий оператор створює петлю у виконанні програми, що приводить до наростання часу виконання всієї програми.

Якщо програмні петлі мають просту структуру, тобто про будь-яку пару петель можна сказати, що вони або не мають загальних елементів, або одна петля вкладена в іншу, то в цьому випадку можна застосувати вищеописані методи аналізу часу виконання.

8. Аналіз програм на псевдомові

Якщо ми знаємо ступінь зростання часу виконання операторів, записаних за допомогою неформальної "людської" мови, то, отже, зможемо проаналізувати програму на псевдомові (такі програми називатимемо псевдопрограмами). Проте часто ми не можемо у принципі знати час виконання тієї частини псевдопрограми, яка ще не повністю реалізована у формальних операторах мови програмування. Наприклад, якщо в псевдопрограмі є неформальна частина, що оперує абстрактними типами даних, то загальний час виконання програми в значній мірі залежить від способу реалізації АТД. Це не є недоліком псевдопрограм, оскільки однією з причин написання програм в термінах АТД є можливість вибору реалізації АТД, зокрема по критерію часу виконання.

При аналізі псевдопрограм, що містять оператори мови програмування і виклики процедур, що мають неформалізовані фрагменти (такі як оператори для роботи з АТД), можна розглядати загальний час виконання програми як функцію поки не певного часу виконання таких процедур. Час виконання цих процедур можна параметризувати за допомогою "розміру" їх аргументів. Так само, як і "розмір вхідних даних", "розмір" аргументів — це предмет для обговорення і дискусій. Часто вибрана математична модель АТД підказує логічно обгрунтований розмір аргументів. Наприклад, якщо АТД будується на основі множин, то часто кількість елементів множини можна прийняти за параметр функції часу виконання.

Ця інформація може використовуватись програмістом для побудови більш ефективної програми.

Часто різниця між поганим і добрим алгоритмом буває більш важлива, чим між швидкодійним і повільним комп’ютером. Приведемо приклад. Нехай потрібно відсортувати масив, який складається з мільйона чисел. Що швидше сортувати його вставками на суперкомп’ютері (100 мільйонів операцій в секунду) або злиттям на домашньому комп’ютері (1 мільйон операцій в секунду)? Нехай сортування вставками написане на асемблері дуже економно, але для сортування  чисел необхідно, скажімо, лише  операцій. В той же час, алгоритм злиттям потребує  операцій.

Для сортування мільйона чисел отримаємо  для суперкомп’ютера і всього  для домашнього комп’ютера.

Ми бачимо, що розробка ефективних алгоритмів не менш важлива, ніж розробка швидкої техніки.

 - аналіз складності отримав широке застосування в багатьох практичних додатках. Але необхідно чітко розуміти його обмеженість.

До основних недоліків можна віднести наступні:

для складних алгоритмів отримання  - оцінок як правило, або дуже трудоємне або практично неможливе;

зазвичай складно визначити складність „в середньому”;

 - оцінка дуже не точна для відображення більш тонких відмінностей алгоритмів;

 - аналіз дає мало інформації для аналізу поведінки алгоритмів при обробці невеликих об’ємів даних.

Класифікація алгоритмів

Постійні – складність не залежить від ;

Лінійні – складність ;

Поліноміальні – складність:, де ;

Екпоненціальні – складність: , де  більша за , а  - поліноміальна функція;

Суперполіноміальні – складність: , де , а  - функція яка зростає швидше за постійну, але повільніше лінійної.

**Приклад виконання лабораторної роботи**

Написати алгоритм, псевдокод та програму на Delphi 7 для обчислення функції F(x,y):

**Блок-схема**

Початок

Введення x та у

Y:=x2+7y

Кінець

Y:=2x+y

Введення результату

Ні

Обчис-лити?

Ні

х<5?

Так

Так

Ні

Вихід?

Так

Рисунок 1 – Приклад блок-схеми алгоритму

**Псевдокод**

**алгоритм** Обчислення функції F (**арг ціле** х, **арг ціле** у, **рез ціле** f)

**початок**

**введення** x, y

**якщо** x<5 **тоді** F:= x\*x+7\*y

**інакше** f:=2\*x+y

**виведення** f

**кінець**

**Примітка.** У псевдокоді базові керуючі структури представляються наступним чином:

* розгалуження if: **якщо** умова **тоді** дія1 **інакше** дія2
* цикл while: **поки** умова **тоді** дія
* цикл for: **для** і від n1 до n2

**Лістинг програми**

unit Unit1;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, ExtCtrls;

type

TForm1 = class(TForm)

Label1: TLabel;

Image1: TImage;

Label2: TLabel;

Label3: TLabel;

Edit1: TEdit;

Edit2: TEdit;

Button1: TButton;

Label4: TLabel;

Edit3: TEdit;

procedure Button1Click(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Form1: TForm1;

implementation

{$R \*.dfm}

//підпрограма натиснення кнопки «Обчислити»

procedure TForm1.Button1Click(Sender: TObject);

var x, y, f: integer; //ініціалізація змінних

begin

// обчислення фунцкії f(x,y)

x:=StrToInt(Edit1.Text); //присвоення змінній х введеного значення

y:=StrToInt(Edit2.Text); //присвоення змінній у введеного значення

if(x<5) then f:=x\*x+7\*y // обчислення f(x,y), якщо x<5

else f:=2\*x+y; // обчислення f(x,y), якщо x>=5

Edit3.Text:=IntToStr(f); // виведення результатів

end;

end.

**Скриншот програми**

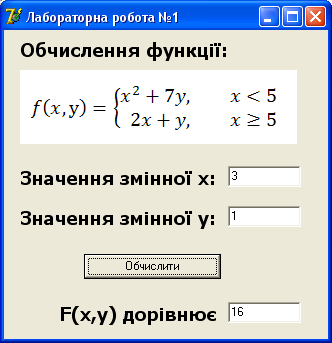


Рисунок 2 ­– Приклад інтерфейсу програми

***Завдання:***

**Постановка задачі**

у відповідності зі своїм варіантом написати блок-схеми, псевдокод алгоритмів та програми (на довільній мові програмування, наприклад, Pascal, C++, Delphi, або будь-якій іншій *структурній* мові програмування) для обчислення вказаних функцій та рішення заданих задач, оцінити їх складність, використовуючи О-символіку.

**Варіант 1**

1. Вводячи в циклі по 4 оцінки, які студенти отримали в сесію, визначити число невстигаючих студентів і середній бал групи по всім екзаменам.

2. Скласти алгоритм визначення найменшого елемента матриці А(m,n), вказавши номер його строки і стовпця.

Варіант 2

1. В області 10 районів. Задані площі, які в кожному році засівають пшеницею, і врожай, який зібраний в кожному районі. Визначити середню врожайність пшениці по кожному району і по області в цілому (за три роки).

2. Зріст учнів в класі представлений у вигляді масиву. Зріст дівчат кодується знаком „+”, зріст хлопців знаком „-”. Визначити середній зріст хлопців.

**Варіант 3**

1. Визначити, скільки різних сигналів може бути подано m прапорцями різних кольорів. Відмінність сигналів заключається в порядку розташування різнокольорових прапорців на щоглі. Розв’язати при m = 6.

2. Нехай маємо ряд спостережень . Скласти алгоритм визначення середнього значення  і середнього квадратичного відхилення  по формулам

; 

**Варіант 4**

1. Біля стінки стоїть нахилена палиця довжиною х. Один її кінець знаходиться на відстані  від стінки. Визначити значення кута  між палицею і підлогою для значень 4,5 м. і , який змінюється від 2 до 3 м. з кроком 0,2 м.

2. Скласти програму для обробки результатів кросу на 500 м. для жінок. В кросі брали участь не більше за 100 учасниць. Для кожної учасниці внесіть її прізвище, шифр групи, прізвище викладача, результат. Потрібно отримати результуючу таблицю, яка впорядкована за результатом, і в якій є інформація про виконані норми 1 розряду. Обчислити сумарну кількість студенток, які виконали цю норму.

**Варіант 5**

1. Скласти таблицю вартості порцій сиру вагою 50, 100, 150, ...,1000 грамів

(ціна 1 кг. – 15 грн.)

2. Прізвища учасників змагань по фігурному ковзанню після короткої програми в порядку, який відповідає зайнятому місцю. Скласти список учасників в порядку їх стартових номерів для вільної програми (учасники виступають в порядку зворотному зайнятим місцям).

**Варіант 6**

1. Почавши тренування, спортсмен в перший день пробіг 10 км. Кожний наступний день він збільшував денну норму на 10 % від норми попереднього дня. Через скільки днів спортсмен буде пробігати в день більше 20 км.

2. Зміни основних фондів галузі описується рівнянням:

.

Заданий план інвестицій  і початкове значення основних фондів . Скласти алгоритм визначення значень .

**Варіант 7**

1. Визначити середній зріст дівчаток і хлопчиків одного класу. В класі навчається n учнів.

2. В пам’яті ПЕОМ зберігаються списки номерів телефонів і прізвища абонентів, впорядковані за номерами телефонів, для кожного з п’яти телефонних вузлів міста. Один телефонний вузол включає декілька АТС (не більше 10). Номери АТС (перші дві цифри номери телефону), які відносяться до кожного телефонного вузла, також зберігаються в пам’яті ПЕОМ. Скласти програму, яка забезпечує швидкий пошук прізвища абонента по заданому номеру телефону.

**Варіант 8**

1. Вводячи в циклі по 5 оцінок кожного студента, отримати число студентів, які не мають оцінок 2 і 3. В групі навчається n студентів.

2. Результати перепису населення зберігаються в пам’яті ПЕОМ. Використовуючи масиви прізвищ і роки народження, надрукувати прізвища і підрахувати загальне число мешканців, які народилися раніше 1930 року.

**Варіант 9**

1. Скласти програму для визначення віку, який підходить в кандидати для вступу в шлюб, використовуючи наступну умову: вік дівчини дорівнює половині віку чоловіка плюс 7, вік чоловіка визначається відповідно як подвоєний вік дівчини мінус 14. Дані для перевірки роботи задачі задати самостійно.

2. Японська радіокомпанія провела опитування 250 радіослухачів по питанню: ”Яку тварину Ви пов’язуєте з Японією і японцями?”. Скласти програму отримання п’яти відповідей, які найбільше зустрічаються і їх часток в (%).

**Варіант 10**

1. Скласти програму, яка реалізує уривок казки: питає: куди бажає піти герой (направо, наліво і прямо), і друкує, що його чекає в кожному випадку. Відповідь ПЕОМ присвоїти символьній змінній і надрукувати. Текст питань і відповідей ПЕОМ задати самостійно.

2. Задані дві вибірки ; . Скласти ефективний алгоритм визначення вибіркового коефіцієнту кореляції:



де ,  - середнє значення вибірок.

 - незміщене середнє квадратичне відхилення спостережень першої вибірки (аналогічно для другої вибірки).

**Варіант 11**

1. Задано n трійок чисел a, b, c. Вводячи їх по черзі і інтерпретуючи як довжини сторін трикутника визначити скільки трійок може бути використано для побудови трикутника (числа a, b, c при введені розташувати в порядку зростання.)

2. Відомо, що в Києві самими теплими є дні з 15 липня по 15 серпня. Для проведення фестивалю були обрані 7 днів підряд найбільш теплих по даним за останні 10 років. Складіть програму для виконання цієї роботи на ПЕОМ.

**Варіант 12**

1. Почавши тренування спортсмен в перший день пробіг 10 км. Кожний наступний день він збільшував денну норму на 10 % від норми попереднього дня. Через скільки днів спортсмен пробіжить сумарний шлях 150 км.

2. При виборі місця будівництва житлового комплексу при металургійному заводі необхідно враховувати „розу вітрів” в даній місцевості. На основі даних щоденного визначення направлення вітру, яке проводилося протягом року, визначити доцільне взаємне розташування промислової і житлової зони.

Вказівка: Напрямок вітру координується наступним чином: 1 – північний, 2 – південний, 3 – східний, 4 – західний, 5 – північно – західний, 6 – північно – східний, 7 – південно – західний, 8 – південно – східний.

**Варіант 13**

1. Визначити, скільки різних сигналів може бути подано m прапорцями різних кольорів. Відмінність сигналів заключається в порядку розташування різнокольорових прапорців на щоглі. Розв’язати при m = 6.

2. Скласти алгоритм визначення найменшого елемента матриці А(m,n), вказавши номер його строки і стовпця.

**Варіант 14**

1. Почавши тренування, спортсмен в перший день пробіг 10 км. Кожний наступний день він збільшував денну норму на 10 % від норми попереднього дня. Через скільки днів спортсмен буде пробігати в день більше 20 км.

2. Визначити середній зріст дівчаток і хлопчиків одного класу. В класі навчається n учнів.

**Варіант 15**

1. Біля стінки стоїть нахилена палиця довжиною х. Один її кінець знаходиться на відстані  від стінки. Визначити значення кута  між палицею і підлогою для значень 4,5 м. і , який змінюється від 2 до 3 м. з кроком 0,2 м.

2. Результати перепису населення зберігаються в пам’яті ПЕОМ. Використовуючи масиви прізвищ і роки народження, надрукувати прізвища і підрахувати загальне число мешканців, які народилися раніше 1930 року.

***Контрольні питання:***

1. Поняття алгоритму, його визначення.

2. Назвіть властивості алгоритму.

3. Яких умов необхідно дотримуватись щоб побудувати алгоритм?

4. Які алгоритмічно розв’язні чи нерозв’язні проблеми відомі на даний час?

5. Як розв’язуються алгоритмічно нерозв’язні проблеми?

6. Які форми представлення алгоритмів Ви знаєте?

7. Які фактори впливають на час виконання програм?

8. Коли і для чого використовують О-символіку?

9. При якому значенні  алгоритм, який потребує  операцій, ефективніший за алгоритм, який потребує  операцій.

10. Знайдіть використовуючи О-символіку, час виконання в найгіршому випадку наступних процедур як функцій від :

а) procegure *matmpy* (n: integer);

var

i, *j, к:* integer; **begin**

**for i: = 1 to *n* do**

**for j:=1 to n do begin**

*C[i,j]:=* 0;

**for *k:=* 1 to *n* do**

*C[i,j]:= C[i,j]* + *A[i,k]* \* *B[k,j]*

**end**

**end**

б) **procegure** *mystery* (*n:* integer);

**var**

i, j, *к:* integer;

**begin**

**for** i:= 1 **to** n - 1 **do**

**for** j:= i + 1 **to** n **do**

**for** *k:=* 1 **to** *j* **do**

{группа операторів з часом виконання 0(1)}

**end**

в) **procegure** *veryodd* (*n -* integer);

**var**

*i, j, x, у:* integer;

**begin**

**for** i:= 1 **to** *n* do

**if** *нечетное(i)* **then begin**

**for** *j:=*1 **to** *n* **do**

*x:* = x + 1 ;

**for** *j:=* 1 **to** i **do**

y:= у + 1

**end**

**end**

\*г) **procegure** *recursive* (n: integer): integer;

**begin**

**if** *n <=* 1 **then**

**return(1)**

**else**

**return** (recursive(n - 1) + *recursive{n -* 1))

**end**

11. Розташуйте наступні функції в порядку зростання а) , б) , в) , г) , д) , е) , ж) , з), и) , к) 17.

12. Дана послідовність чисел . Покажіть, що за час  можна визначити, чи є в цій послідовності два однакові числа.

13. Дано масив S який складається з  дійсних чисел, а також число . Як за час  визначити, чи можна представити  у вигляді суми двох елементів масиву S?