ZAJĘCIA NR 6

3 metody „rozpracowania" algorytmu.

1) Metoda wstępująca.
Aby napisać algorytm – siadamy i piszemy go linia-po-linii. Jesteśmy w stanie ta zrobić, bo jest to zaledwie prosty problemik, „pestka" nawet dla jednego programisty.

2) Metoda zstępująca
Tutaj z kolei, mając złożony problem, musimy najpierw rozłożyć go na kilka prostszych problemów i być może rozdzielić je na kilku programistów. Każdy z nich z kolei dalej będzie rozkładać problemy na-co-raz-to-mniej-sze aż w końcu dojdziemy w ten sposób do pojedynczych instrukcji.

3) Metoda zstępująco-wstępującą.
Jest ona połączeniem dwu poprzednich metod. Najpierw – gdy mamy jeszcze jeden złożony problem – stosując metodę zstępującą, rozkładając go tym samym na mniejsze i prostsze problemy. Dopiero wówczas, gdy jesteśmy w stanie rozwiązać je już „od ręki" – w ten właśnie sposób je rozwiązujemy (a więc metodą wstępującą). Prowadzenie metody zstępującej przerywamy więc w tym momencie, kiedy jej kontynuacja byłaby już nieefektywna. Odpowiednie zestawienie w tej metodzie dwóch poprzednich, czyni ją najlepszą – najprostszą i najbardziej efektywną.

Jak dobrze pisać programy?

Zauważmy, że prawie zawsze, gdy coś piszemy – robimy to kierując swe słowa do kogoś innego. Pisząc staramy się ująć temat prosto i przedstawić w zgodzie z regułami gramatycznymi, jednak nie ze względu na owe reguły, lecz ze względu na łatwość i szybkość zrozumienia go przez przyszłego czytelnika. Rozmijanie się z tematem i pogmatwany styl świadczą o braku szacunku dla czytelnika.

Jedna z faz pracy nad programem związana jest właśnie z jego napisaniem (inne – to na przykład: rozeznanie tematu, przemyślenie rozwiązania, zgromadzenie środków finansowych itp.). Proste „programiki" piszą pojedynczy programiści. Większe jednak projekty wymagają pracy zbiorowej. W takiej sytuacji, gdy wymagana jest współpraca, jeszcze bardziej trzeba zwracać uwagę na poprawny styl pisania. Dodatkowo uzmysławia nam to, że styl programowania, to nie tylko zestaw reguł, których należy ślepo przestrzegać, ale – przede wszystkim – wyrażenie doświadczenia kontaktu osobistego.

Podczas pisania programu należy zwracać uwagę na trzy poniżej podane aspekty.

1) Pierwszym z nich jest oczywiście opinia człowieka, który będzie korzystać z tego programu. Użytkownik programu – podobnie jak czytelnik – chce, aby kierowana do niego informacja przedstawiona była w jasnej i łatwo przyswajalnej formie. Całkiem nierozsądnym jest żądanie od użytkownika rozumienia źle sformułowanych fraz. Owe frazy pojawiają się, gdy użytkownik jest proszony o podanie danych (są to tzw. „komunikaty zachęcające") czy informowany o wyniku (cząstkowym czy końcowym) działania programu. Poprawne sformułowanie owych fraz zmniejsza prawdopodobieństwo ich nieprawidłowej interpretacji.

2) Drugim – koniecznym do uwzględnienia – aspektem jest inny programista, który będzie chciał zapoznać się pisanym przez nas programem. Może on chcieć zapoznać się z cudzym programem ze względu na cały szereg przyczyn, np. aby wykorzystać w swoim programie algorytm, przygotowany przez nas, albo aby adoptować nasz program na swoje potrzeby. W dowolnym z tych przypadków, będzie on wdzięczny autorowi, który napisał zrozumiały i poprawnie ułożony program.

3) Trzecim aspektem, który należy brać pod uwagę przy pisaniu programu, jest jego autor. Nawet jeśli nikt inny nie będzie wykorzystywać czy analizować naszego programu, wysiłki, ukierunkowane na osiągnięcie specyficznej lekkości i zrozumiałości programu, będą procentować na każdym kroku. Należy od samego początku nauczyć się dobrego stylu programowania i później już tylko „dotrzymywać mu kroku". Dzięki temu podczas pracy nad bardziej złożonymi programami, ów programista już nie będzie musiał przezwyciężać wcześniej pokonanych ograniczeń. Wiadomo, „czym skorupka za młodu…".

Jakie cechy są przynależne, a jakie nie, dobrym programistom

Choć metodyka programowania w znacznym stopniu zbudowana jest na zdrowym rozsądku, niektóre jej elementy są o tyle istotne, że wymagają formalizacji. Rady, co należy robić a czego należy unikać, nie służą bynajmniej do wstawiania programisty w "wąskie ramki". Zasady te pomogą natomiast pisać mu bardziej poprawne programy. Jeśli – choćby jak podaliśmy powyżej wychodząc od zdrowego rozsądku – będziemy dostatecznie wypełniać te zalecenia, to wówczas program będzie łatwiej opisać i wykorzystywać.

1) Należy wykorzystywać identyfikatory znaczące (nazwy mnemotechniczne) – tj. takie które bezpośrednio odnoszą się do tego co opisują (np. na pomoc „pom", a nie np. u),
2) Należy wykorzystywać stopniowe formy pisania programu , tj. pisać go metodą zstępującą lub zstępująco-wstępującą („dziel i zwyciężaj"!)
3) Nie należy wykorzystywać jednej i tej samej zmiennej dla wielu celów (mogło by to prowadzić do nieporozumień),
4) Nie należy umieszczać wielu instrukcji w jednej linii; (co linia, to instrukcja; złożoną instrukcję pisze się nieraz nawet w wielu liniach; dla przejrzystości tekstu stosuje się w nim liczne wcięcia)
5) Nie należy „gonić" za efektywnością (za wszelką cenę),
6) Wszelkie teksty kierowane do użytkownika muszą być pisane zrozumiałym (jasnym i prostym) językiem,
7) W tekście programu należy zamieszczać komentarze.

Rozpatrzmy kilka algorytmów.

1) na rozwiązywanie równania kwadratowego
2) na wygenerowanie „tabliczki mnożenia”
3) ma porządkowanie szeregu liczb.

AD 1)

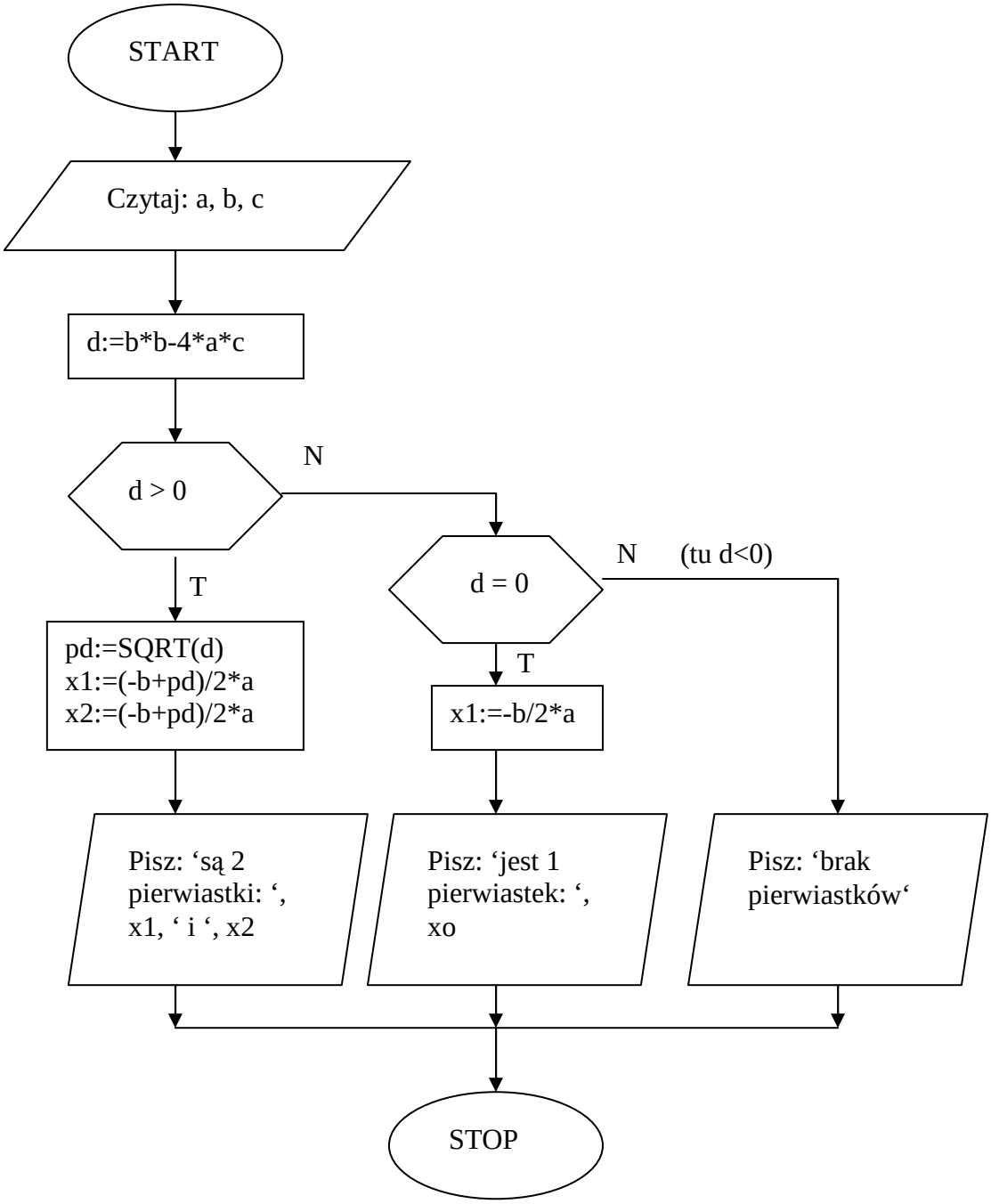
Jest to jedne z najprostszych algorytmów. W jego rozpisaniu opieramy się na zasadach matematycznych rozwiązywania równania kwadratowego:

1. Mamy równanie $ax^2+bx+c=0$.
2. Liczymy deltę $\Delta=b^2-4ac$.
3. Gdy $\Delta>0$ – to są 2 pierwiastki (rozwiązania) równe: $x_{1,2}=\frac{-b\pm\sqrt{\Delta}}{2a}$
4. Gdy $\Delta=0$ – to jest 1 pierwiastki (podwójny) równy: $x_0=\frac{-b}{2a}$
5. Gdy $\Delta<0$ – to brak jest pierwiastków (równanie nie ma rozwiązania na zbiorze liczb rzeczywistych – jak to się ładnie mówi)

Rysując schemat uwzględnimy, że:

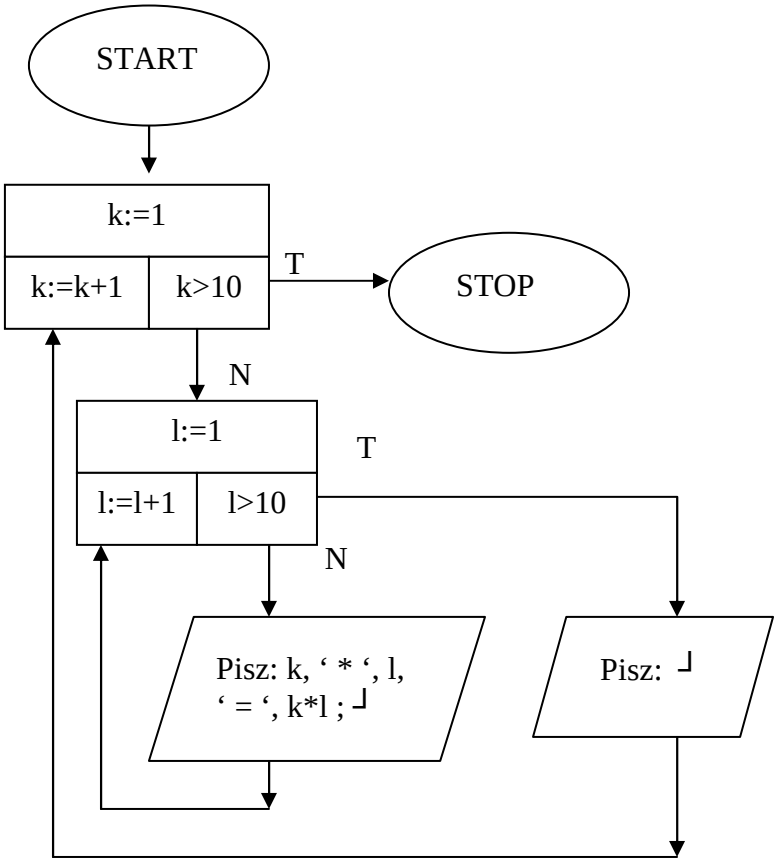
1) Równanie kwadratowe w jednoznaczny sposób jest opisane jedynie przez jego współczynniki - stąd już tylko one wystarczą nam do rozwiązania równia kwadratowego.
2) Deltę zamiast za pomocą niedopuszczalnego w językach programowania symbolu Δ, oznaczać będziemy symbolem d.
3) Jako że nie można stosować indeksów (dolnych czy górnych, sprowadzimy je do głównej linii – zamiast x_1 napiszemy więc x1.
4) Pierwiastek kwadratowy oddamy za pomocą funkcji SQRT – tak więc zamiast $\sqrt{\Delta}$ napiszemy (zważywszy dodatkowo na punkt 2), SQRT(d)
5) Ponieważ niedopuszczalne jest pisanie „słupkowe” wzorów – pierwiastki równania oddamy liniowo (np. pisząc: x1:=-b/2*a).
6) komputer pisząc coś – pisze to literalnie (tj. tak-jak-jest-podane) gdy tekst ten jest ujęty w apostrof, a podaje wartości zmiennych, gdy nie ujęte są w apostrof. Przy tym:
   - poszczególne fragmenty tekstu (a więc ujęte i nie ujęte w apostrof) należy oddzielać przecinkami.
   - należy pamiętać o spacjach (odstępach) i przecinkach (w tekście).

7) jako że liczenie pierwiastka kwadratowego jest dla komputera skomplikowaną operacją – warto obliczyć go tylko raz – przed podstawieniem do wzorów na x1 i x2 (jest pamiętamy pod zmienną pd – pierwiastek z delty)


START
Czytaj: a, b, c
d:=b*b-4*a*c
d > 0
N
T
d = 0
N
(tu d<0)
T
pd:=SQRT(d)
x1:=(-b+pd)/2*a
x2:=(-b+pd)/2*a
x1:=-b/2*a
Pisz: ‘są 2 pierwiastki: ‘, x1, ‘ i ‘, x2
Pisz: ‘jest 1 pierwiastek: ‘, xo
Pisz: ‘brak pierwiastków‘
STOP

AD 2)
Będziemy wypisywać „tabliczkę mnożenia” – 10 słupków po 10 mnożeń

W tym przypadku:
1) zastosujemy 2 zmienne : k – będzie oznaczać mnożną, a l – mnożnik
2) co wypiszemy linię – to przejdziemy do nowej linii (ozn.:┘ - jak ENTER), a dodatkowe zastosowanie tego symbolu po danej kolumnie będzie znów oznaczało przejście do nowej linii, co w efekcie da nam jedną pustą linię (odstęp między kolumnami)
3) żadne dane zewnętrze nie muszą tu być wczytywane.



Wynik działania tego programu będzie następujący:

1*1=1
1*2=2
…
1*10=10

2*1=2
2*2=4
…
2*10=20

…

10*1=10
10*2=20
…
10*10=100

AD 3)

Wyobraźmy sobie ciąg liczb, np.: 5, 7, 2, 6, 1, 12.
Chcemy uporządkować wyrazy tego ciągu od najmniejszego do największego.
Możemy wyobrazić sobie tę sytuację z analogiczną – układaniem otrzymanych z rozdania kart od najmniejszej do największej (tam się to chyba nazywa od najmłodszej do najstarszej).

a) metoda przestawieniowa

1. Znajdź najmniejszy wyraz w ciągu i PRZESTAW go z pierwszym (jeśli najmniejszy nie jest już na I pozycji). W ten sposób na I pozycji jest wyraz najmniejszy.
2. Znajdź najmniejszy wyraz wśród wyrazów jeszcze nieuporządkowanych (tj. od pozycji 2 do ostatniej) i jeśli wyraz ten nie jest na II pozycji – przestaw go z wyrazem na II pozycji. W ten sposób i II element będzie na swojej pozycji.
3. Opisaną w poprzednim punkcie procedurę powtarzaj do wyrazu przedostatniego (ostatniego nie trzeba już brać tu pod uwagę, bo spośród niego i następnych /których nie ma!/ jest już na pewno najmniejszy!).

Zobaczmy, jak on działa na przytoczonym tu (powyżej) przykładzie.
Aby za dużo nie pisać:
- szarym kolorem oznaczamy wyraz na pozycji na którą zaraz będziemy wstawiać najmniejszy wyraz spośród jeszcze nieuporządkowanych,
- żółtym – najmniejszy z pozostałych (z nim właśnie go przestawimy),
- zielnym – wyrazy już uporządkowane.

5, 7, 2, 12, 1, 6.
1, 7, 2, 12, 5, 6.
1, 2, 7, 12, 5, 6.
1, 2, 5, 12, 7, 6.
1, 2, 5, 6, 7, 12. – nie przestawialiśmy
1, 2, 5, 6, 7, 12. – nie przestawialiśmy (ostatniego z przedostatnim), a ostatniego też nie przestawiamy (bo DLA OSTATNIEGO nie mamy już z którym!)

b) metoda wstawieniowa
Głównie tę metodą wykorzystuje się w porządkowaniu kart.

1. Znajdź najmniejszy wyraz w ciągu i WSTAW go na pierwszą pozycję (jeśli najmniejszy nie jest już na I pozycji). W ten sposób na I pozycji jest wyraz najmniejszy.
2. Znajdź najmniejszy wyraz wśród wyrazów jeszcze nieuporządkowanych (tj. od pozycji 2 do ostatniej) i jeśli wyraz ten nie jest na II pozycji – wstaw go na II pozycję. W ten sposób i II element będzie na swojej pozycji.
3. Opisaną w poprzednim punkcie procedurę powtarzaj do wyrazu przedostatniego (ostatniego nie trzeba już brać tu pod uwagę, bo spośród niego i następnych /których nie ma!/ jest już na pewno najmniejszy!).

Zobaczmy, jak on działa na przytoczonym tu (powyżej) przykładzie.
Aby za dużo nie pisać:
- żółtym kolorem oznaczamy najmniejszy wyraz z jeszcze rozpatrywanych,
- zielnym – wyrazy już uporządkowane.

5, 7, 2, 12, 1, 6.
1, 5, 7, 2, 12, 6.
1, 2, 5, 7, 12, 6.
1, 2, 5, 7, 12, 6.
1, 2, 5, 6, 7, 12. – nie przestawialiśmy
1, 2, 5, 6, 7, 12. – nie przestawialiśmy (ostatniego z przedostatnim), a ostatniego też nie przestawiamy (bo DLA OSTATNIEGO nie mamy już z którym!)

b) metoda bąbelkowa

1. Bierz 2 kolejne wyrazy (1. i 2., 2. i 3., 3. i 4., …, n-1. i n.) i przestawiaj je miejscami „jeśli trzeba” (tj. gdy większy jest przed mniejszym)
2. Przebiegi j.w. robimy tak długo, aż w pewnym momencie w wyniku któregoś z nich nic nie zmienimy.

Zobaczmy, jak on działa na przytoczonym tu (powyżej) przykładzie.
Aby za dużo nie pisać:
- zielonym kolorem oznaczamy te (sąsiednie) 2 elementy, które dopiero co przestawiliśmy,
- pustym wierszem oddzielamy kolejne przebiegi.

5, 7, 2, 12, 1, 6. – od tego układu zaczynamy
5, 2, 7, 12, 1, 6.
5, 2, 7, 1, 12, 6.
5, 2, 7, 1, 6, 12.

2, 5, 7, 1, 6, 12.
2, 5, 1, 7, 6, 12.
2, 5, 1, 6, 7, 12.

Widzimy, że w ten sposób powolutku małe przesuwają się na przód, a duże na tył.

2, 1, 5, 6, 7, 12.

1, 2, 5, 6, 7, 12.

1, 2, 5, 6, 7, 12. po tym „bezproduktywnym” przebiegu kończymy (bo nie ma już większego przed mniejszym).

Najprostsza z tych metod (dla komputera) jest metoda przedstawieniowa.

Zastanówmy się „dlaczego?”.
Ciąg n zmiennych w komputerze pamiętany jest w tzw. tablicy jednowymiarowej o długości n. Stanowi ja ciąg n pól w których – kolejno – stoją poszczególne elementy rozważanego ciągu.

Tabela t:

5	7	2	12	1	6

Poszczególne pola tabeli oznacza się ujmując po jej nazwie ich numer w nawiasie kwadratowym. Stąd też w powyższym przykładzie: t[1]=5, t[2]=7, …, t[6]=1.
Zobaczmy najpierw jak można zamienić wartościami 2 zmienne.
Niech np. x=3 i y =5. Chcemy, by było na odwrót (tj. x=5 i y=3). Jak to zrobić? (oczywiście operujemy na wartościach , które – hipotetycznie nie znamy, stąd nie możemy powiedzieć: było x=3 i y =5, więc teraz jest na odwrót, tj. x=5 i y=3).
Owego przestawienia nie możemy dokonać wykonując: x:=y i y:=x. Wtedy bowiem (na naszych przykładowych danych) otrzymalibyśmy: po podstawieniu x:=y → x=5 (a więc teraz tak x, jak i y jest równe 5, a co za tym idzie zagubiliśmy dotychczasową wartość x /=3/), a więc podstawienie y:=x nic nie zmieni. Sytuację tę obrazuje poniższa tabela.

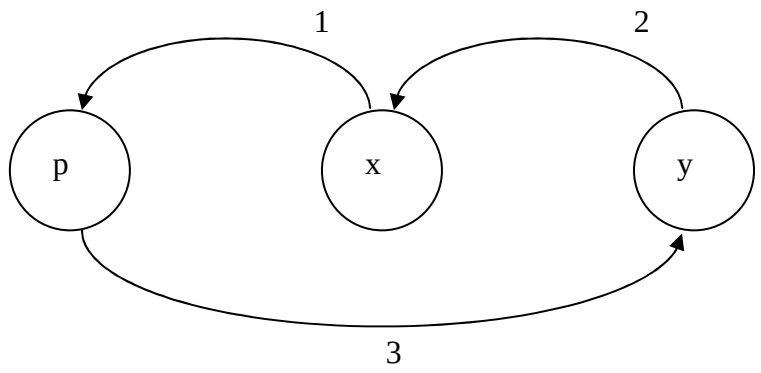
	x	y
na początku	3	5
po x:=y	5	5
po y:=x	5	5

Musimy więc jakoś to x zapamiętać w zmiennej pomocniczej (np. p).

	x	y	p
na początku	3	5	
po p:=x	3	5	3
po x:=y	5	5	3
po y:=x	5	3	3

W tabelach tych stany początkowe x i y zaznaczono zielonym kolorem, a stany końcowe tych zmiennych – kolorem czerwonym. Widzimy, że tylko w drugiej z tych tabel dokonaliśmy przestawienia wartości.

Jeszcze graficzne to przedstawmy (numery przy strzałkach oddają kolejność przypisań):



Jak więc przestawić w tabeli t 1. i 4. jej pozycję (tj. 5-kę z 1-ką)?

Tabela t:

5	7	2	12	1	6

Analogicznie! (traktując t[1] jak x, a t[5] jak y):
p:=t[1]
t[1]:=t[5]
t[5]:=p

Umiejąc przestawiać poradzimy sobie z rozwiązaniem a) i z rozwiązaniem c)
W przypadku rozwiązania a) mamy jednak mniej przestawień.
Porównajmy więc je jeszcze pod względem skomplikowania (dla komputera) z rozwiązaniem b).
W jego przypadku, aby WSTAWIĆ element z pozycji k na pozycję l (k>l), co w kartach jest proste, tu wygląda bardziej skomplikowanie.

Przykład:

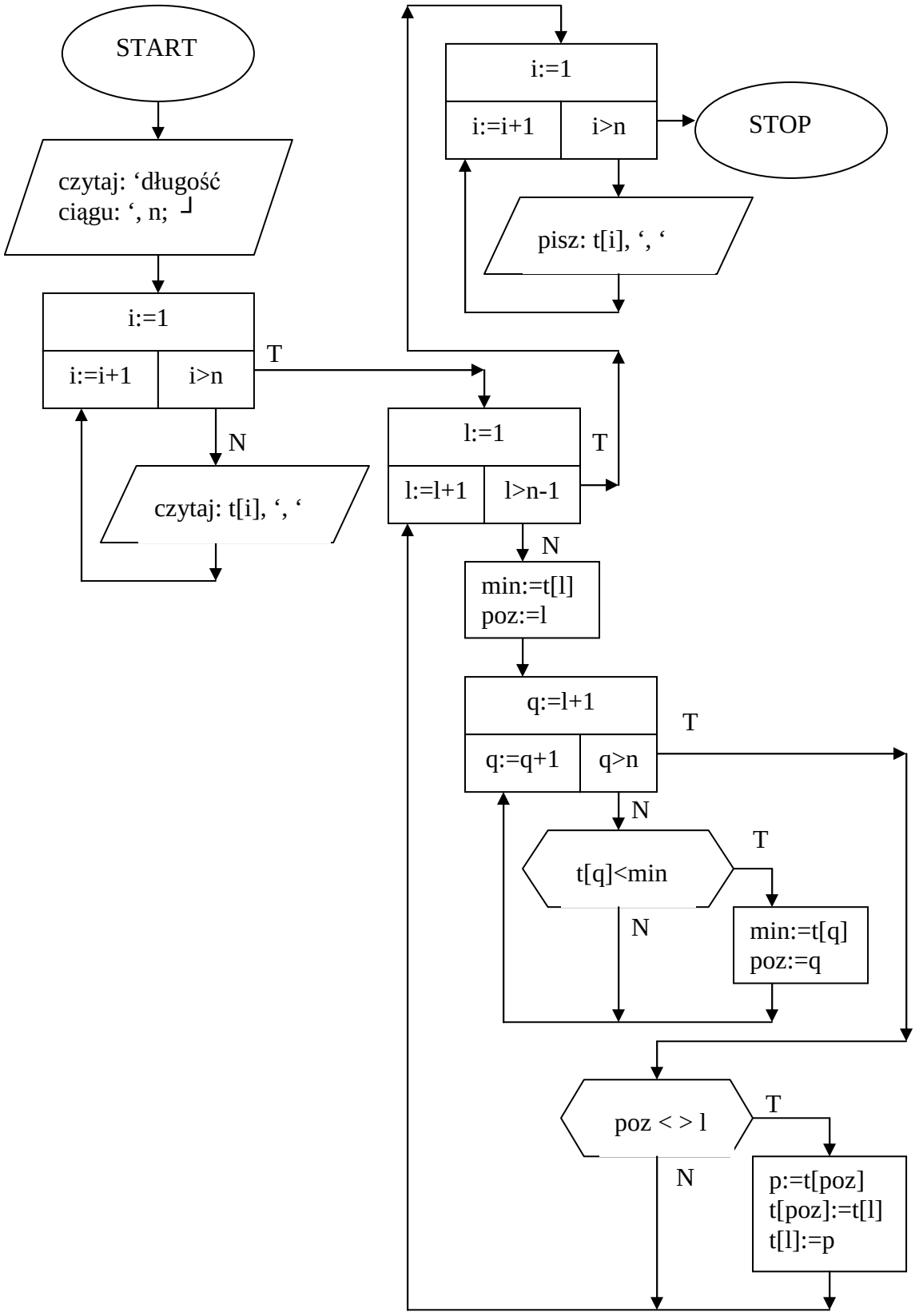
pozycja	1	2	3(l)	4	5	6(k)	7
Przed ciągiem operacji	2	4	10	15	8	7	12
Po tym ciągu operacji	2	4	7	10	15	8	12

Widzimy, że w tym przypadku musimy:
- wiedzieć które są już na swoim miejscu (pierwsze dwa, tj. od pozycji 1 do l-1),
- następnie wśród pozostałych poszukać najmniejszy (tu: „7” na pozycji k=2),
- potem zapamiętać tę „7” w zmiennej pomocniczej (np. p),
- skopiować zawartości kolumn: 5-ej w 6-ą, 4-ej w 5-ą, 3-ej w 4-ą (tj. z pozycji k-a downto l odpowiednio na pozycje k downto l+1),
- i w końcu ową „7” wstawić w pozycję l (tu: l=3).

W ten sposób wybraliśmy już metodę – przez przestawianie! (tj. a)).
Opiszmy ją szerzej.
Poniżej przedstawiamy jego schemat blokowy.

Przy okazji zauważmy, że:

1) to, co już wcześniej opracowaliśmy – tu możemy bez ponownego zagłębiania się w sprawę od razu zastosować,
2) ostatnia skrzynka decyzyjna zawiera symbol „< >”, który w informatyce oznacza „różne od”
3) w skrzynce tej ów symbol można zastąpić symbolem „>”, bo jeśli pojawi się coś mniejsze od minimum, to na pewno po prawej stronie od wcześniej ustalonego.


START
czytaj: ‘długość ciągu: ‘, n; ┘
i:=1
i:=i+1
i>n
T
N
czytaj: t[i], ‘, ‘
l:=1
l:=l+1
l>n-1
T
N
min:=t[l]
poz:=l
q:=l+1
q:=q+1
q>n
T
N
t[q]<min
T
N
min:=t[q]
poz:=q
poz < > l
T
N
p:=t[poz]
t[poz]:=t[l]
t[l]:=p
i:=1
i:=i+1
i>n
STOP
pisz: t[i], ‘, ‘

C.D.N. …